UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

Langtons maur og logiske porter

Martin Dang

Masteroppgave, våren 2016



Langtons maur og logiske porter

Martin Dang

2. mai 2016

Forord

Jeg vil først og fremst takke mine dyktige veiledere, Roger Antonsen og Herman Ruge Jervell, som har introdusert meg til denne oppgaven. De har gitt meg god veiledning som gjorde at jeg kunne fullføre oppgaven. Takk for at dere har hatt tro på prosjektet mitt.

Jeg vil også takke mamma og pappa for all støtten og maten de har laget til meg. Dere er verdens beste.

Takk til mine venner og medstudenter som har tatt seg tid til å hjelpe meg gjennom studiet hver gang jeg er i knipe.

Jeg har valgt å skrive denne oppgaven på norsk fordi det er det språket jeg føler meg komfortabel å skrive i. Noen ord har også blitt fornorsket mens de fleste engelske ordene lot jeg være.

Sammendrag

Denne oppgaven handler om å undersøke atferden til Langtons maur og hvordan vi kan bruke maurens atferd til å konstruere logiske porter. Videre skal vi bruke de logiske portene til å utføre beregninger. Senere skal vi se på en mer generell variant av Langtons maur og se om vi kan konstruere logiske porter på samme måte som vi gjorde for Langtons maur.

Innhold

Figurer

Tabeller

Kapittel 1

Introduksjon

Langtons maur er en virtuell maur funnet opp av Christopher Langton. Christopher Langton er en forsker innenfor informatikk og er grunnleggeren av fagfeltet som kalles *kunstig liv* [9]. En av modellene som blir brukt til å simulere kunstig liv er *cellulære automater*. Cellulære automater har blitt mye brukt i forskjellige fagfelter for å modellere ulike fenomener. Cellulære automater ble først utviklet av Stanislaw Ulam og John von Neumann. De brukte cellulære automater til å simulere selvproduserende organismer [12]. Langtons maur er en virtuell maur som lever på et todimensjonal brett. Mauren beveger seg etter to enkle regler som gir en interessant atferd. Vi tar i denne oppgaven sikte på å studere atferden til denne mauren. Vi skal utnytte atferden til mauren til å representere *logiske porter*.

1.1 Motivasjon

Denne oppgaven er inspirert av artikkelen til Gajardo [4]. Denne artikkelen handler blant annet om logiske porter med Langtons maur. I [4] finner dere konstruksjoner som vi i denne oppgaven skal bygge på. Vi skal lage nye porter og vi skal se på hvor begrensede disse portene er. Videre skal vi bruke de logiske portene vi konstruerte til å lage en krets. Vi skal bruke cellulære automater som modeller til å visualisere hvordan vi kan lage logiske porter med Langtons maur. Det siste vi skal gjøre er å se om det går an å utvide og konstruere logiske porter med den generelle Langtons maur ved å bruke den samme teknikken som Gajardo beskrev i [4].

1.2 Kapitteloversikt

I kapittel 2 skal vi gi en oversikt over bakgrunnstoffet som vi har hatt nytte av i oppgaven. Det blir gitt en kort introduksjon om bakgrunnen til cellulære automater før vi skal se på hva en cellulær automat er. Vi skal også gå gjennom grunnleggende boolsk algebra og sammenhengen mellom boolsk algebra og logiske porter. Boolsk algebra og logiske porter skal vi senere bruke Langtons maur til å representere.

I kapittel 3 skal vi se på tidligere resultater som vi skal bruke til å bygge videre på i de neste kapitlene. Vi skal se på et resultat om maurens atferd i seksjonen *ubundet spor*, som får oss til å tenke på hvordan mauren kommer til å utvikle seg gjennom over tid.

I kapittel 4 skal vi gå grundig gjennom hva som ble brukt i artikkelen [4] for å få en bedre forståelse om hvordan vi skal konstruere logiske porter. Vi skal blant annet se på hvordan vi representerer logiske porter med Langtons maur.

I kapittel 5 skal vi vise hvordan man lager logiske porter med Langtons maur og bruke de logiske portene som vi konstruerte til å representere kretser. Vi skal se på hvilken porter som kan konstrueres og hvilken som ikke kan konstrueres.

I kapittel 6 skal vi utvide Langtons maur til den generelle Langtons maur og se om vi kan utføre de samme operasjonene som vi gjorde i kapittel 5 med den generelle Langtons maur.

Kapittel 2

Innledning

I dette kapitlet gis det en kort oversikt over automatteori og turingmaskiner før vi skal grundigere gå gjennom hva en cellulær automat er. I tillegg skal vi se på grunnleggende boolsk algebra, operasjoner og kretser som vi kommer til å ha bruk for senere.

2.1 Kunstig liv

Kunstig liv er et fagfelt hvor man undersøker biologiske fenomener som er relatert til liv og hvordan de utvikler seg over tid. Datamaskinen har vært en stor hjelp i det å konstruere slike modeller. Det finnes tre hovedtyper av kunstig liv: *soft, hard og wet* type. *Hard* type består av å bruke hardware-basert kunstig liv. Roboter og droner som utfører arbeid på egenhånd, er gode eksempler på kunstig liv av *hard* type. *Wet* type av kunstig liv er den biokjemiske kunstig liv delen som baserer seg på biologi hvor man lager synteser av blant annet DNA. Den delen av kunstig liv som er relevant for denne oppgaven er *soft* type, som baserer seg på software, hvor man simulerer et liv-lignende system gjennom datamaskinen eller andre digitale medier. Teknikker som blir brukt i *soft* type er blant annet nevrale nettverk og cellulære automater, som vi skal se på hva er senere [9]. Nevrale nettverk knyttes gjerne til en maskinlæringsalgoritme som blir "lært" opp av inputverdier og utvikler seg videre ut fra det (les mer i [10]). Kunstig liv kan vurderes fra ulike filosofiske perspektiver og synsvinkler og oppfattes forskjellig. Noen er mer ekstreme enn andre. Vi har svak (ikke det samme som soft) kunstig liv, og de bruker kunstig liv til å lære mer om biologisk liv ved å konstruere modeller og andre prosesser assosiert med levende organismer gjennom simuleringer av datamaskiner og andre digitale

medier. I motsetning til sterk kunstig liv, og de som mener at det er en realisering av liv [8].

2.2 Automater

Før vi skal se på hva cellulære automater er skal vi gå fort gjennom grunnleggende automatteori. Automater er abstrakte modeller av maskiner som utfører beregninger på en input ved å bevege seg gjennom en eller flere tilstander på en diskret måte. En *tilstand* er en teknisk term for all informasjon som er lagret i et gitt tidspunkt. Det finnes også en *transisjonsfunksjon* som beregner hva den neste tilstanden skal være avhengig av inputen. En transisjonsfunksjon er en funksjon som har to argumenter: den nåværende tilstanden og en inputstreng i *alfabetet*, og den returnerer en ny tilstand. Inputstrengen til en automat er en sekvens av symboler fra alfabetet. Et alfabet er en endelig mengde av symboler. Det finnes fire hovedtyper av automater [1]:

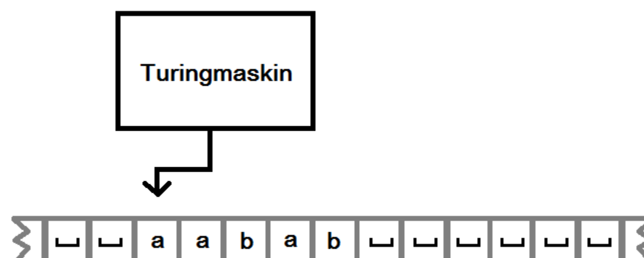
- Endelige tilstandsmaskiner.
- Pushdown-automater.
- Lineært bundne automater
- Turingmaskiner

De fire hovedtypene av automater er rangert etter hva de kan beregne. Endelige tilstandsautomater er de enkleste, mens turingmaskiner er de mest komplekse og kan beregne mye mer.

2.3 Turingmaskiner

I denne oppgaven er turingmaskinen den mest relevante automaten. Turingmaskinen var først foreslått av Alan Turing i 1936 [7]. Den er veldig mye lik endelige tilstandsmaskiner, bortsett fra at den har ubegrenset med minne. En turingmaskin kan beregne alt en dagens datamaskin kan beregne. Turingmaskinen bruker en uendelig lang *tape* som sitt minne. Turingmaskin har også en *hode* som den bruker til å lese eller skrive på tapen og kan bevege seg fram og tilbake på tapen. Turingmaskinen starter beregningen med inputstrengen på tapen og det blanke symbolet på alle de andre stedene.

Eksempel 2.3.1. I figur 2.1 ser vi en turingmaskin med inputstrengen *aabab*. På tapen kan vi se inputstrengen og det blanke symbolet ␣.

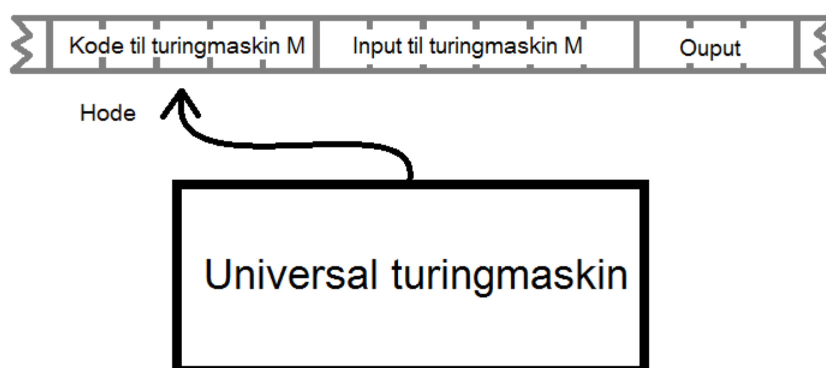


Figur 2.1: En turingmaskin med inputstreng aabab på tapen.

Hvis turingmaskinen trenger å lagre informasjon, kan den gjøre det ved å skrive på tapen. Vi kan tenke oss at turingmaskinen har en tilstand på tapen og en indre tilstand hvor den indre tilstanden er som en hjerne.

2.3.1 Universell turingmaskin

En universell turingmaskin U er en turingmaskin som tar to inputer som argument. Den tar inn en turingmaskin T og en inputstreng w og simulerer w på turingmaskinen T. Outputen til T etter å ha blitt simulert med input w blir skrevet ut på tapen til U.

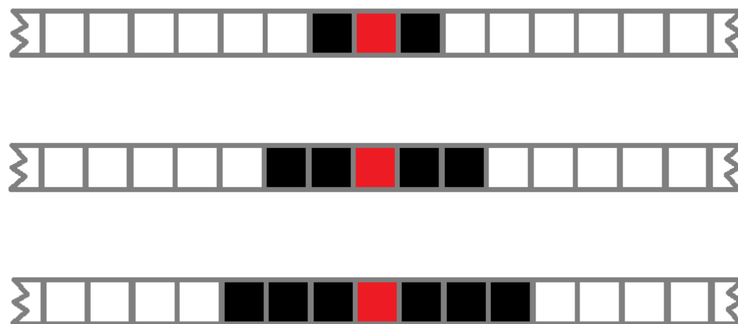


Figur 2.2: Figur av en universell turingmaskin.

2.4 Cellulære automater

Cellulære automater er en diskret modell og kan visualiseres i et rutenett, hvor hver rute kalles en *celle* [12]. Hver celle i rutenettet har en endelig antall mengder med tilstander. Ofte er hver tilstand visualisert med en bestemt farge. Tilstandene i cellen blir oppdatert samtidig i diskret tid etter en forhåndsdefinert regel, kalt *regelstreng* [12]. En cellulær automat har tre hovedegenskaper: dimensjon, antall tilstander per celle og antall naboceller. Dimensjonen i cellulær automaten beskriver hvilken dimensjon rutenettet har. Antall tilstander er et tall k som er større eller lik 2. *Naboceller* er en mengde med celler rundt en celle i som er med på å bestemme den neste tilstanden til cellen i i den neste *generasjon* [12]. Når tiden t øker med 1, kaller vi tid $t + 1$ for den neste generasjonen.

Eksempel 2.4.1. I figur 2.3 er det tre eksempler på nabocellen til en endimensjonal cellulær automat hvor den røde cellen representerer cellen i og de svarte cellene er i sin nabocelle.

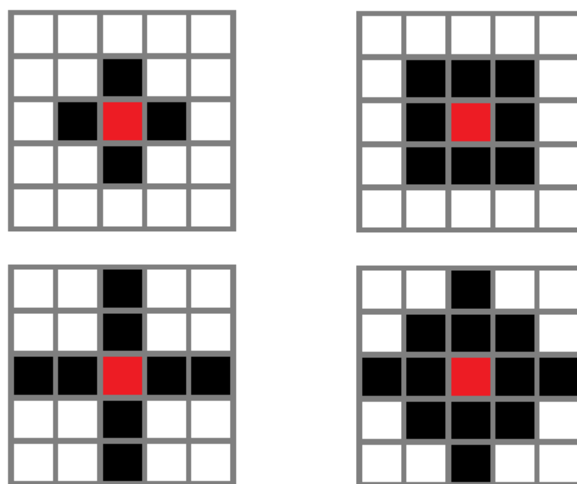


Figur 2.3: Eksempler på naboceller til endimensjonale cellulære automater.

I figur 2.4 finnes det fire andre eksempler for todimensjonale cellulære automater. Jo høyere i dimensjonen man kommer, kan man definere nabocellene i flere forskjellige former.

2.4.1 Endimensjonale cellulære automater

Den enkleste cellulære automaten er den endimensjonale cellulære automaten som består av en uendelig lang tape med celler. Og vi ser

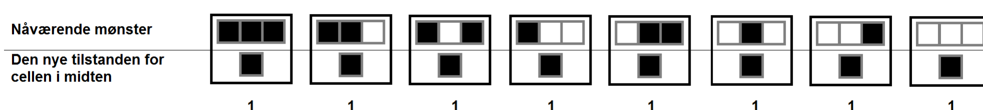


Figur 2.4: Eksempler på naboceller til todimensjonale cellulære automater.

på kun den minste mengden med naboceller rundt hver celle i tapen, nemlig en nabo på hver side. Og hver celle har to tilstander. Vi kaller den endimensjonale cellulære automaten for den minste ikke-trivielle cellulære automaten. For å gjøre det enkelt sier vi at de to tilstandene er svart og hvit. Og hele tapen blir initialisert til hvit. En cellulær automat tar inn en beskrivelse om hvordan den cellulære automaten skal utvikle seg i de neste generasjonene. Denne beskrivelsen kaller vi en *regelstreng*. En regelstreng for en endimensjonal cellulær automat blir representert som en 8-bits tall.

Eksempel 2.4.2. Her ser vi et eksempel på en regel for en endimensjonal cellulær automat i figur 2.5. I dette tilfellet settes hver kombinasjon av nåværende mønster til svart. Måten vi representerer 8-bits regelstreng tallet er når den nye tilstanden for cellen i midten settes til svart legges det til en 1 og hvit legges det til en 0. I figur 2.5 settes alle den nye tilstanden til svart og regelstrengen blir 11111111 som tilsvarer 255 i titallssystemet.

Siden det er åtte mulige kombinasjoner med nåværende mønster og to mulige tilstander den nye cellen kan ha i den neste generasjonen, gir det $2^8 = 256$ mulige regelstrenger. Dette er for en endimensjonal cellulær automat med $k = 2$ og hvor mengden av naboceller er 1 på hver side. Vi ser antall regelstrenger øker eksponentielt ved å øke antall



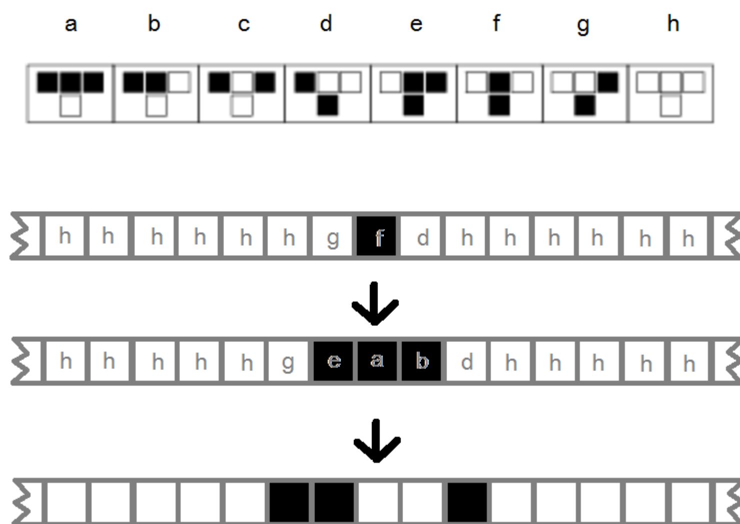
Figur 2.5: Eksempel på en regelstreng for en endimensjonal cellulær automat.

nabocellen eller antall tilstander per celle. For en endimensjonal cellulær automat med 3 tilstander og 1 nabocelle på hver side gir det $3^3 = 27$ kombinasjoner av nåværende mønster og $3^{27} = 7625597484987$ antall regelstrenger.

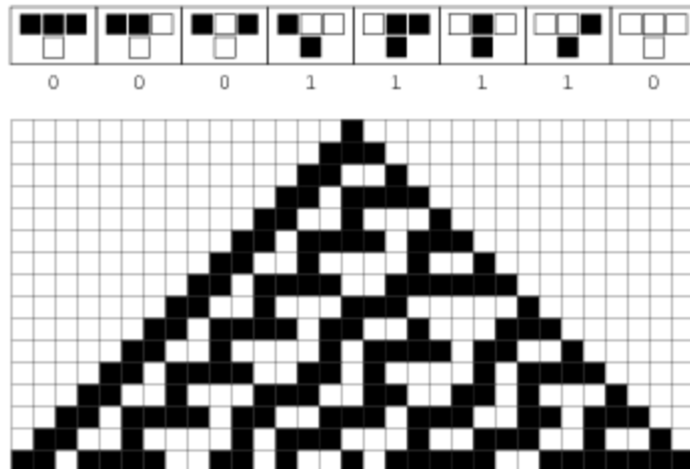
Eksempel 2.4.3. Her er to eksempler, i figur 2.7 og 2.8, på endimensjonal cellulære automater med regelstrenger 00011110 og 10010110 som starter med en svart tilstand på tapen. Måten vi representerer dette på er at vi legger tapen med den neste generasjonen nedover hverandre. Som vi ser er det kun en svart tilstand i den første raden. På andre rad har vi den andre generasjonen hvor regelstrengen har blitt utført på raden som er et hakk ovenfor (første raden). Slik fortsetter det nedover. La oss se på endringene på de tre første generasjonene for å få et bilde om hvordan vi bruker regelstrengen. I figur 2.6 har vi markert regelstrengen med bokstaver slik at det blir lettere å se hvilken regel på nåværende mønster som blir brukt på hver celle. På hver celle i tapen, er det markert med grå bokstaver hvilken nåværende mønster cellen har.

I eksempel 2.7 og 2.8 så vi på endimensjonale cellulære automater hvor initialtilstanden er én svart tilstand. I boka *A New Kind of Science* [18] eksperimenterte forfatteren Wolfram med cellulære automater. Wolfram simulerte alle 256 regelstrengene til endimensjonal cellulær automater. Men han simulerte også tilfeldige initialtilstander. Det vil si at det ikke var en svart tilstand på tapen, men opp til flere svarte tilstander på tapen og i forskjellige mønstre. Wolfram oppdaget at han kunne klassifisere de endimensjonale cellulære automatene i fire klasser [18]:

- Klasse 1: Den cellulære automaten utvikler seg til et stabilt mønster. Og når den starter på et tilfeldig brett utvikler nesten alle mønstrene til samme slutttilstanden.
- Klasse 2: Den cellulære automaten utvikler seg til en periodisk mønster. Det kan være mange forskjellige slutttilstander, men alle



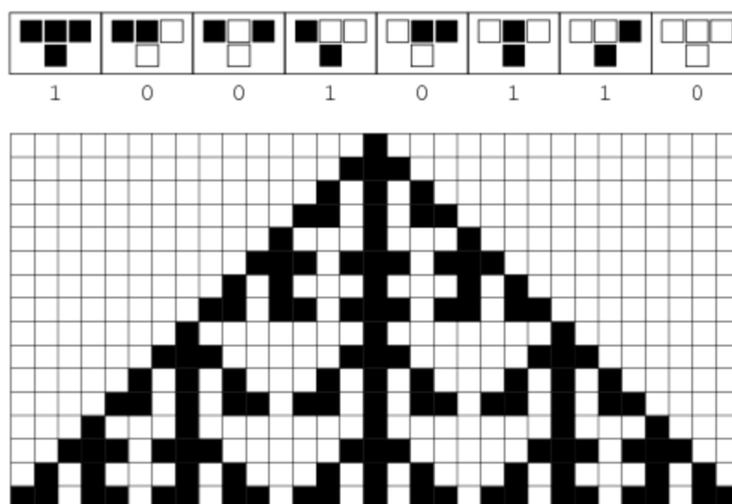
Figur 2.6: Utvikling av en endimensjonal cellulær automat for regelstreng 00011110 med en svart celle på rutenettet.



Figur 2.7: Endimensjonal cellulær automat med regelstreng 00011110.

slutttilstandene er enkle strukturer som enten repeterer for alltid eller repeterer etter et visst antall steg.

- Klasse 3: Den cellulære automaten utvikler seg kaotisk.

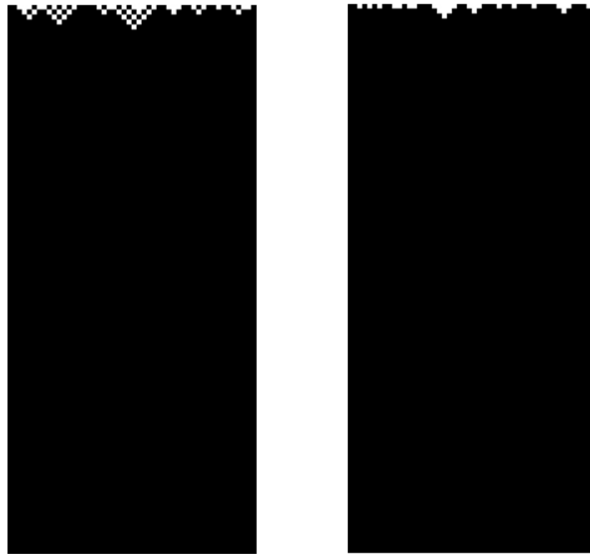


Figur 2.8: Endimensjonal cellulær automat med regelstreng 10010110.

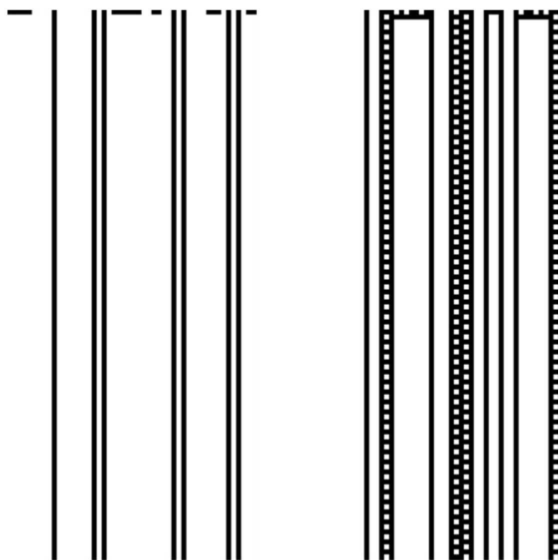
- Klasse 4: Den cellulære automaten utvikler seg til en blanding av ordnede og tilfeldige mønstre. Det oppstår lokale mønstre som kommuniserer med hverandre på en komplisert måte.

Disse klassene kan sammenlignes med molekyler i fastform, væske og gass. I begynnelsen definerte Wolfram klassene ut fra hvordan mønstrene så ut, men senere da flere detaljerte egenskaper ble kjent viste det seg at det var en sammenheng mellom klassene og egenskapene[18].

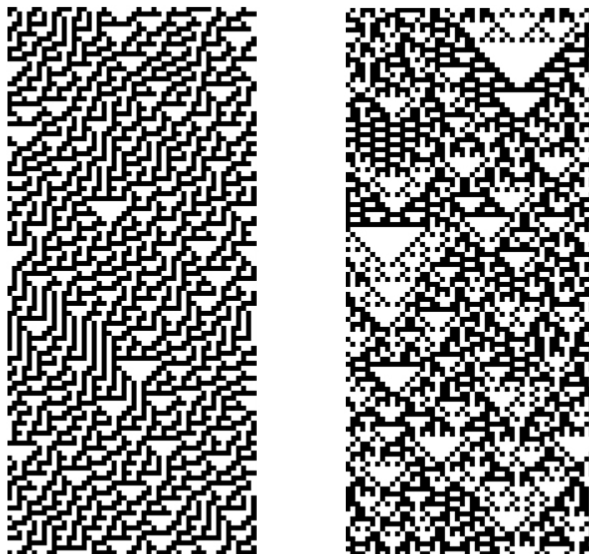
Eksempel 2.4.4. Vi skal i dette eksempelet se på cellulære automater i de forskjellige klassene [2]. I figur 2.9 er et eksempel av to regelstrenger til en endimensjonal cellulær automat som er i klasse 1. Vi kan se at det blir fort stabilt og alle tilstander ender likt (i disse tilfellene svart). I figur 2.10 kan vi se endimensjonale cellulære automater i klasse 2 med enkle periodiske strukturer som blir repetert. I figur 2.11 er eksempler på klasse 3 hvor det er kaotisk og det finnes ikke noe periodiske mønstre. I figur 2.12 er eksempler på endimensjonale cellulære automater i klasse 4. Mønstrene er komplekse og strukturen former seg lokalt i rutenettet.



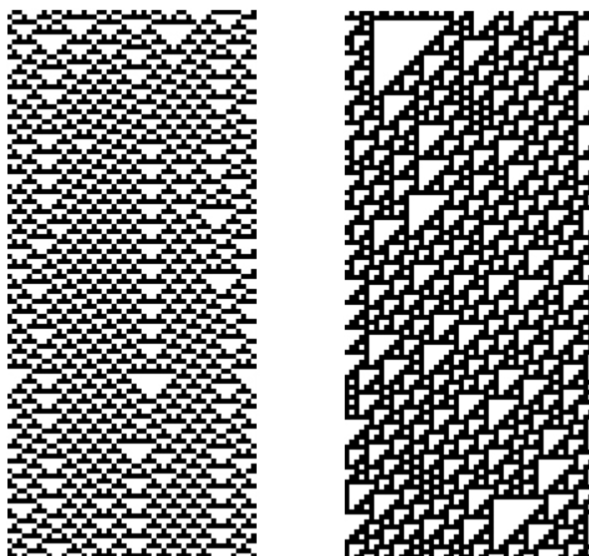
Figur 2.9: Regelstreng 11111010 og 11111110.



Figur 2.10: Regelstreng 00000100 og 01101100.



Figur 2.11: Regelstreng 00011110 og 01011010.



Figur 2.12: Regelstreng 00110110 og 01101110.

2.5 Boolsk algebra og porter

Boolsk algebra er algebra hvor hver variabel har to verdier. Disse verdiene er som oftest kjent som SANN og USANN, eller 1 og 0. Boolsk algebra brukes blant annet til å beskrive digitale kretser og har tre hovedfunksjoner: OG, ELLER og IKKE. Andre funksjoner er XOR, NAND, NOR og XNOR som kan avledes med OG, ELLER og IKKE. En funksjon kan beskrives med en funksjonsuttrykk eller sannhetsverditabell. Nedenfor i figur 2.13 kan man se hvordan sannhetsverditabellen til de tre hovedfunksjonene ser ut og under tabellen kan man se funksjonsuttrykkene, hvor $\vee, \wedge$ og $\neg$ tilsvarer ELLER, OG og IKKE. Sannhetsverditabellen er en tabell som består av alle mulige kombinasjoner av 1 og 0 for variablene [11].

A	B	OG
1	1	1
1	0	0
0	1	0
0	0	0

A ^ B

A	B	ELLER
1	1	1
1	0	1
0	1	1
0	0	0

A v B

A	IKKE
1	0
0	1

¬A

Figur 2.13: Sannhetsverditabell for OG, ELLER og IKKE.

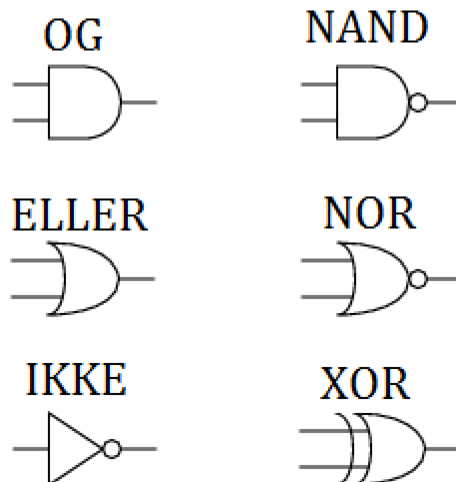
2.5.1 Regneregler

Vanlige regneregler for boolsk algebra [11]:

- $A \vee B = B \vee A$
  og $A \wedge B = B \wedge A$ (kommutativ)
- $A \vee (B \vee C) = (A \vee B) \vee C$
  og $A \wedge (B \wedge C) = (A \wedge B) \wedge C$ (assosiativ)
- $A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$
  og $A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$ (distributiv)

En *logisk port* er den grunnleggende byggesteinen i alle digitale systemer. Logiske porter tar inn en eller flere inputer og sender ut en output. Med logiske porter kan vi uttrykke boolske funksjoner.

I figur 2.14 ser vi hvordan de mest vanlige portene er representert (les mer [16]).



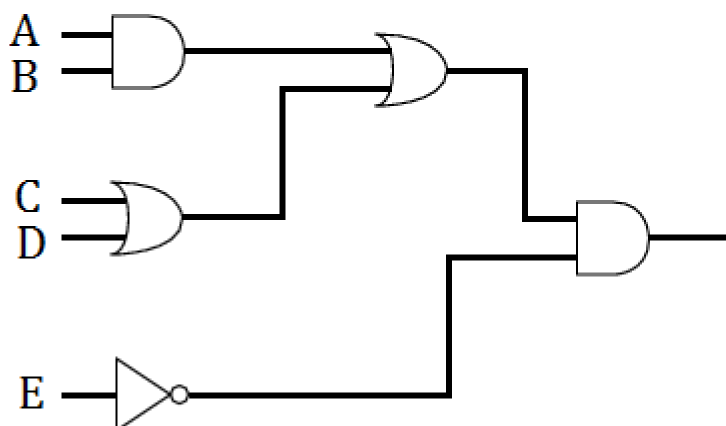
Figur 2.14: Hvordan vi representerer de forskjellige logiske portene.

Vi kan tenke oss at det er en bryter som slår på strøm i inputledningene. Hvis vi ser på OG-porten, har den to inputledninger fra venstre siden og en outputledning ut fra høyre side. For de to inputledningene finnes det fire forskjellige kombinasjoner av strøm som går inn i porten. OG-porten sender ut strøm kun hvis den får inn to ledninger med bryteren på (strøm i ledningen). Det er en sammenheng mellom boolsk algebra og logiske porter. Hvis vi ser på sannhetsverditabellen for OG ser vi alle kombinasjonene for 1 og 0. Vi kan relatere dette med strøm som kommer inn i en input eller ikke.

2.6 Kretser

Kretser er samlinger av porter som er koblet sammen. Kretser kan modellere boolske funksjoner.

Eksempel 2.6.1. I figur 2.15 er et eksempel på en liten krets med fem inputer. Kretsen tilsvarer den boolske funksjonsuttrykket
$((A \wedge B) \vee (C \vee D)) \wedge \neg E)$



Figur 2.15: Eksempel 2.6.1.

2.7 Game of Life

En av de mest kjente cellulære automatene er Game of Life laget av John Horton Conway [13]. Game of Life er en todimensjonal cellulær automat som skal forestille celler i bevegelse. Hver celle har to tilstander representert med fargene svart og hvit. Nabocellene til en celle *i* er definert som de åtte cellene rundt *i* slik som det andre bildet i figur 2.4. Reglene til hvordan cellene endrer tilstand er følgende:

- Celler med færre enn to naboceller med svart tilstand blir hvit.
- Celler med svart tilstand og to eller tre svarte naboceller forblir svart til den neste generasjonen.
- Celler med svart tilstand og har mer enn tre svarte naboceller blir hvit.
- Celler med hvit tilstand som har akkurat tre svarte naboceller bytter til svart tilstand.

Dette visualiseres ikke på samme måte som en endimensjonal cellulær automat ved å sette tapene nedover hverandre. I dette tilfelle legges ikke rutenettet over hverandre, men vi ser på kun rutenettet for en generasjon og rutenettet oppdateres på en todimensjonal cellulær automat for hver generasjon. Ved å fargelegge tilstand 0 hvit og tilstand 1 svart, ser

vi når vi simulerer dette på en datamaskin at det ikke er uvanlig å bruke metaforen at det er "liv" i Game of Life hvor svart tilstand er en levende celle og hvit tilstand er en ikke-levende celle. Ved å studere og eksperimentere med Game of Life oppdager man at det finnes mange mønster som kan dannes under en simulasjon. Mønstrene er delt inn i tre grupper: statiske mønster, repeterende mønster og mønster som blir repetert bortover i en retning på rutenettet. Game of Life er ikke bare interessant og fascinerende, men det viser seg å være like sterk som en universell turingmaskin[6]. I artikkelen [6] konstruerte Paul Rendell en turingmaskin ved å bruke mønstrene til å blant annet konstruere logiske porter i Game of Life. Med mønstrene konstruerte Rendell en universell Turingmaskin. Det er gjerne opp til "oppfinneren" av den cellulære automaten hvordan man skal visualisere hans eller hennes cellulære automat. I boka [18] finnes det todimensjonale cellulære automater som blir visualisert ved å legge rutenettene opp på hverandre.

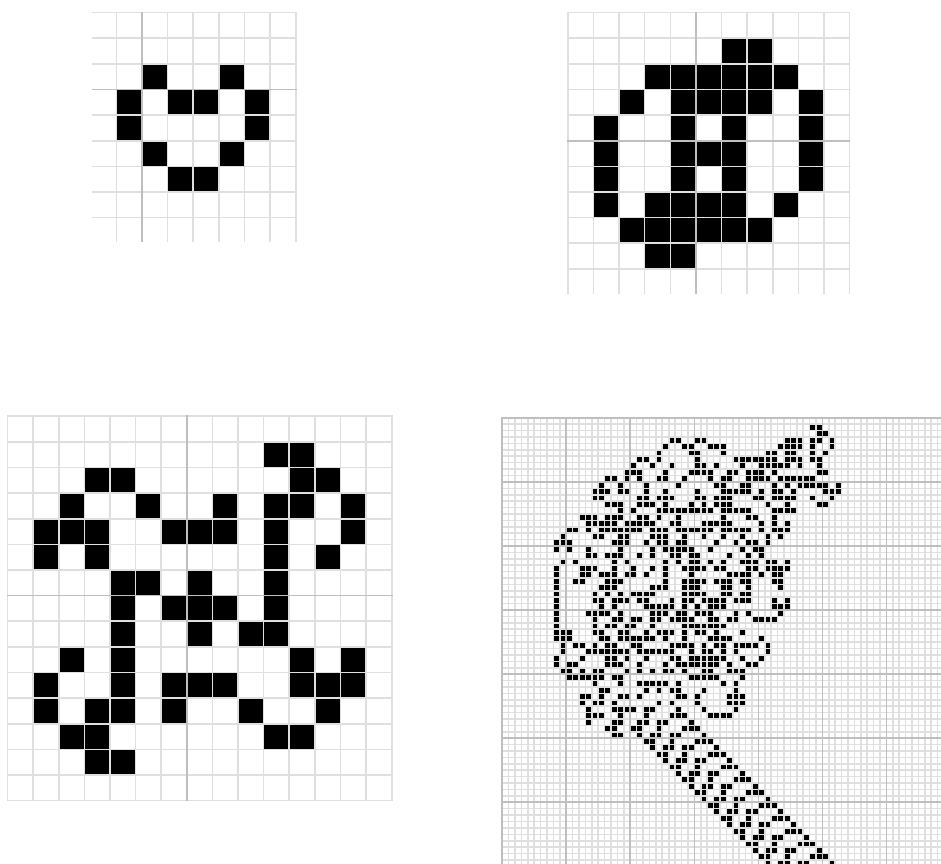
Nå har vi sett på endimensjonale cellulære automater og Conways Game of Life. Det finnes uendelig mange cellulære automater. Det er bare fantasien som setter grenser. Man kan lage cellulære automater i flere dimensjoner eller lage regler som ikke bare ser på de nærmeste naboene men de nærmeste k nabo. Det neste vi skal se på ligner mer på Game of Life enn endimensjonale cellulære automater.

2.8 Langtons maur

Hovedfokuset i denne oppgaven er en cellulær automat oppfunnet av Christopher Langton i 1986 kalt Langtons maur [15]. Langtons maur er en todimensjonal cellulær automat. Langtons maur er en virtuell maur som går rundt i et uendelig stort todimensjonalt rutenett, hvor bevegelsen til mauren er bestemt av et sett med regler. Dette kan bli visualisert i et todimensjonalt rutenett hvor alle celler blir initialisert til hvit og mauren starter på et hvilken som helst celle. Siden det kun er en maur i automaten, er det kun en celle som bytter farge for hver generasjon. Mauren følger to veldig enkle regler:

- Hvis mauren er på en hvit celle, snur den seg 90 grader til høyre, bytter cellen den står på og går deretter et skritt framover.
- Det andre tilfellet er hvis mauren står på en svart celle. Da skal den snu seg 90 grader til venstre, bytte farge på den cellen den står på og gå et skritt framover.

Bare ved å følge de to reglene ser det ut som mauren følger et slags mønster. Deretter ser det ut som mauren beveger seg i tilfeldig kaos, helt til den når steg 10000 og danner en *highway* [15]. En highway er når mauren repeterer et endelig antall sekvens med bevegelser uendelig mange ganger. Det er ikke sikkert man ser ved første øyekast at Langtons maur er en todimensjonal turingmaskin. Langtons maur sitt alfabet er svart og hvit og de eneste tilstandene mauren har er de fire retningene nord, vest, sør og øst. Dette gjør at mauren ikke har noe annet minne enn retningen sin. Det Langtons maur gjør er å lese inn en input fra en celle i, skrive ut output på celle i og bevege seg til neste celle som er en av nabocellene til celle i. Det er nøyaktig det en turingmaskin gjør: leser fra tapen, skriver ut på tapen og beveger hodet.



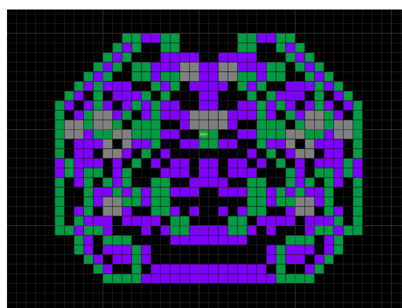
Figur 2.16: Bilder av Langtons maur etter steg 51, 184, 368 og over 10000.

I figur 2.16 ser vi rutenettet til Langtons maur som starter på et blank brett på steg 51, 184, 368 og over 10000. I de tre første figurene ser vi at mauren utvikler seg til symmetriske mønstre. Etter omtrent 10000 steg

lager mauren den såkalte highway i retning mot nedre høyre hjørne og fortsetter evig den retningen.

Senere ble det presentert en mer generell variant av Langtons maur med flere tilstander i hver celle hvor hver farge representerer en tilstand. Den generelle versjonen tar inn en regelstreng som beskriver reglene til mauren. Regelstrengen til den generelle Langtons maur består av en sekvens med R og L. L betyr at mauren skal snu 90 grader til venstre (mot klokken) og gå rett frem, og R betyr at mauren skal snu seg 90 grader til høyre (med klokken) og gå rett fram. Antall farger er lengden på regelstrengen. Når en maur skal besøke en celle med farge *i*, hvor *i* er en indeks fra *0* til *k-1* der *k* er lengden av regelstrengen, får cellen fargen *(i+1)(mod k)* etter at mauren har besøkt cellen og beveget seg etter L eller R som den leste på indeks *i* på regelstrengen. Det vil si at alle celler bytter farge i samme rekkefølge.

Eksempel 2.8.1. Anta regelstrengen er "LLRR" da blir k = 4. For enkelthets skyld sier vi: farge 1 er svart, farge 2 er grønn, farge 3 er lilla og farge 4 er grå. Etter en stund likner *sporet* i figuren på det som kalles en kardioide. Kardioide er en kurve i planet som har en hjerte-aktig form. Sporet til en maur er mengden av celler som mauren har besøkt. Hvis man lar mauren gå lenger vil kardioiden blir større og større.



Figur 2.17: Den generelle Langtons maur med regelstreng LLRR som input.

Todimensjonale rutenett er en fin måte å visualisere cellulære automater på. I resten av oppgaven holder vi til denne modellen og kommer til å bruke todimensjonale rutenett i de fleste figurer og eksempler.

Kapittel 3

Tidligere arbeid

Langtons maur er mindre kjent enn Game of Life og få resultater finnes om atferden til mauren. Ett av de få resultatene som har blitt publisert sier at sporene til mauren er ubundet som ble vist i [5]. Den andre artikkelen vi skal se på er [4] som er den artikkelen vi skal bygge resten av oppgaven på.

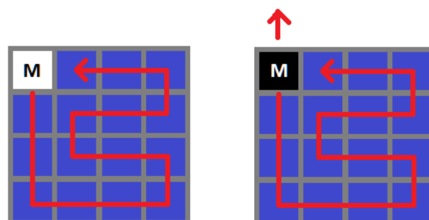
3.1 Ubundet spor

Noe av det man observerer er at sporene til Langtons maur utvider seg mer og mer. Dette ble bevist og er veldig godt forklart i *The mathematical intelligencer* av David Gale [5].

Theorem 3.1.1. *Mauren sitt spor er alltid ubundet hvis det er minst én L og én R i regelstrengen.*

Bevis: Første ting å legge merke til er at vi kan alltid vite hvilken celle en maur kommer fra ved å se på et brett hvor generasjonen er større enn 1. Dette medfører at vi kan spole tilbake til tidligere generasjoner og se hvordan mauren har beveget seg. Hvis mauren sitt spor hadde vært begrenset innenfor et område, da måtte mønstrene på sporet til mauren eventuelt bli repetert. På grunn av den første observasjonen vil sporet være periodisk og alle cellene i sporet ville ha blitt besøkt uendelig mange ganger. En annen ting å merke seg er at mauren bevege seg vertikal og horisontal annen hver gang. Det gjør at vi kan alltid dele alle cellene inn i enten en *h-celle* eller en *v-celle* slik at det brettet kan deles inn i to partisjoner. En h-celle er en celle som mauren kan komme inn og ut

fra venstre og høyre mens en v-celle er en celle hvor mauren kan komme inn og ut ovenfra og nedenfra.



Figur 3.1: Illustrasjon av bundet spor.

Vi skal vise at sporet til mauren ikke kan bli repetert flere ganger innenfor et område. Anta nå at sporet til mauren er bundet da vil det finnes en *maksimal* celle *M*, som er en hjørnecelle i sporet hvor to naboceller aldri blir besøkt. Anta at *M* er en h-celle i sporet som blir repetert og cellen ovenfor og til venstre for *M* har aldri blitt besøkt slik vi ser i figur 3.1. Det betyr at mauren må besøke cellen fra venstre og kommer ut av cellen nedenfra. Dette medfører at tilstanden til celle *M* økes med en. Vi vet at hvis det finnes minst én R i regelstrengen, vil en av tilstandene fra 0 til *k-1* være en tilstand hvor mauren svinger til høyre. Dette fører til, på en diskret tidspunkt så vil *M* ha den tilstanden slik at mauren gå oppover og utenfor det bunnende sporet. Fordi tilstanden til alle cellene endrer seg etter formelen *(i+1)(mod k)*, hvor *i* er nåværende tilstand og *k* er lengden av regelstrengen. Den andre antakelsen er hvis *M* er en v-celle. Dette kan vi vise på samme måte som vi viste for *M* er en h-celle.

Dette er en viktig egenskap som kan man kan få nytte av, som vi kan ha i bakhodet når vi skal konstruere ting med Langtons maur. Teoremet sier at sporet til mauren er ubundet og man vet ikke om en maur besøker andre steder før sporet utvider seg. Det fører til at noen regelstrenger for en maur utvider seg fortere enn andre regelstrenger. Den andre viktige bemerkningen er at rutenettet er delt opp i h-celler og v-celler noe vi senere må ta hensyn til når vi skal konstrure logiske porter til mauren. Vi kan tenke oss at h-cellene og v-cellene er delt opp som et uendelig stor sjakk brett hvor svarte felter er h-celler og hvite felter er v-celler eller motsatt. Dette begrenser en maur så den ikke kan komme inn og ut i en celle fra alle kanter, men bare 2 av kantene.

3.2 Universell maur

I artikkelen *Complexity of Langton's ant* [4] representerte Gajardo og gjengen en konstruksjon av boolske porter med en maur. De boolske portene brukte de til å simulere en endimensjonal cellulær automat. Dette resulterte i at konstruksjonen er P-hard og at systemet kan utføre universelle beregninger. P-hard er en kompleksitetsklasse (les mer [14]). Når vi sier at et system kan utføre universelle beregninger, mener vi at systemet kan utføre alle algoritmer. Systemets inputdata og outputdata må være definert slik at systemet kan utføre algoritmen. I dette tilfellet med Langtons maur, er input og output definert som to celler ved siden av hverandre på et bestemt rutenett. I de neste kapitelene skal vi utvide det Gajardo og gjengen har konstruert.

3.3 Sammendrag

I dette kapittelet har vi sett forskning som har blitt gjort om Langtons maur. Oppgaven vår baserer seg på en utvidelse av artikkelen *Complexity of Langton's ant* [4]. Artikkelen er veldig kort og presis. Noe av det kan være litt uforståelig i første øyekast. I det neste kapittelet skal vi utdype det som har blitt beskrevet i [4].

Kapittel 4

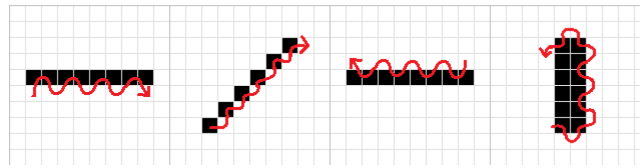
Langtons maur og porter

I kapittel 2 så vi på eksempelet med kardiodiden og den vanlige Langtons maur (Langtons maur med regelstrengen LR). De fleste som har hørt om Langtons maur forbinder det med en maur som starter med et tomt brett og følger et sett med regler slik at det skjer masse endringer av farger på brettet. I dette kapitlet skal vi se på noen viktige konstruksjoner som er grunnmuren til de logiske portene. For å konstruere de logiske portene initialiserer vi brettet med bestemte tilstander slik at vi kan kontrollere bevegelsen til mauren. Ved å kontrollere bevegelsen til mauren kan vi få mauren til å bevege seg og lese en input og skrive outputen på et bestemt sted i rutenettet.

4.1 Sti

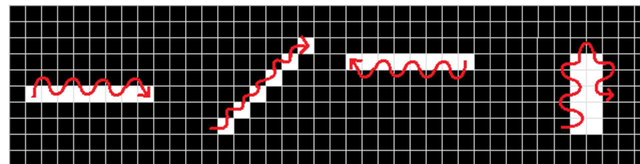
For å kontrollerer bevegelsen til mauren må vi lage en sti som mauren kan bevege seg etter. Stien skal vi bruke til å lede mauren til inputen og outputen. Stien til mauren er en viktig konstruksjon som også brukes til å koble til innganger og utganger på *porter* og *krysninger* som vi skal definere i de neste seksjonene. Måten vi skal lage stien til mauren på er å la den bevege seg som en slange. Når en slange skal bevege seg i en retning, beveger den seg sikksakk i den retningen. Siden mauren ikke kan gå rett fram, men svinger til høyre eller venstre, kan vi bruke slangeidéen til å få mauren til å bevege seg i en bestemt retning.

I figur 4.1 ser vi hvordan vi kan konstruere en sti til mauren hvor vi bruker de svarte cellene som sti på et hvitt rutenett. Vi kan også definere stien på en annen måte. Vi kan også la initialbrettet starte med fargen svart og bruke hvite celler som sti. Forskjellen er at når vi simulerer dette



Figur 4.1: Svart sti med hvit bakgrunn.

vil mauren være på motsatt side av stien. En ting å legge merke til er at når mauren har gått på stien, vil ikke stien se ut som den gjorde før mauren gikk på den. Tilstandene på stien vil bytte på samme måte som det vanlige eksempelet med Langtons maur vi definerte på slutten av kapittel 2.



Figur 4.2: Hvit sti med svart bakgrunn.

I eksemplene framover kommer vi til å demonstrere konstruksjonen vår der initialbrettet har hvite celler og stien til mauren har svarte celler.

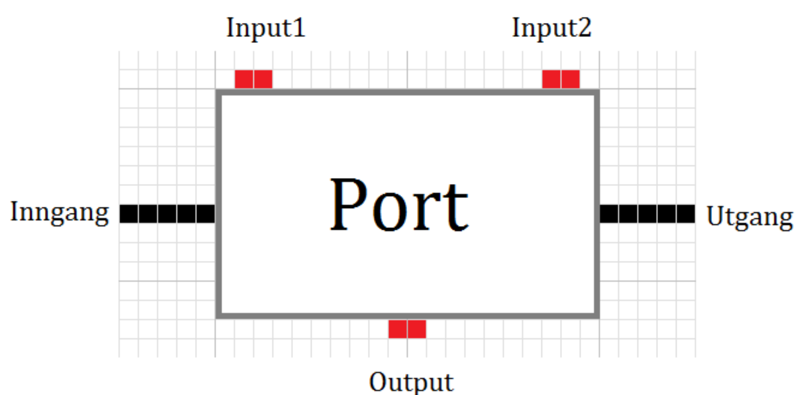
4.2 Input og output

I logiske porter finnes input og output i form av ledninger som kan sende signaler. I den vanlige Langtons maur skal vi definere inputen og outputen på en annen måte. Inputen og outputen skal representeres som to celler ved siden av hverandre hvor enten begge cellene er svarte eller hvite, der svart tilsvarer SANN og hvit USANN. Det er definert på forhånd hvor disse input- og outputcellene skal ligge på rutenettet. I alle porter som vi skal se på, har outputen standard farge som hvit. Mauren fungerer som en "budbringer" hvor den ser på en eller flere inputer og deretter outputer en verdi. Det som skjer er at mauren sjekker ut en eller flere inputer og deretter lar mauren enten outputen være hvit og går videre eller så følger mauren stien ned og endrer outputen til svart før den går videre ut fra hvilken logisk port det er. Inputen og outputen kan ligge så langt fra hverandre som de vil fordi vi kan strekke stien så langt som vi vil, men har en begrensning på hvor nærme hverandre de kan

ligge. Dette er fordi krysninger vi bruker har en fast størrelse som ikke kan bli mindre.

4.3 Port med Langtons maur

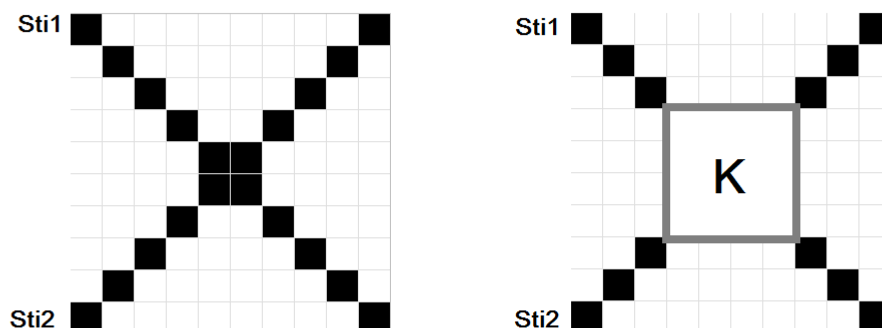
Vi har definert stien, inputen og outputen til portene vår. Portene vi konstruerer kommer til å se slik ut, som i figur 4.3. Vi har valgt å la inputen og outputen være i de samme posisjonene som Gajardo lagde sine porter. Inputene kommer ovenfra og outputen nedenfra. Det som skjer er at mauren kommer inn i stien ved inngangen, leser inputdataene og skriver ut riktig output. Når det er gjort, beveger mauren seg videre og ut utgangen. I figur 4.3 har porten to inputer, men portene kan ha alt fra 1 til n porter, hvor n er et naturligtall.



Figur 4.3: Idéen om hvordan en port skal se ut.

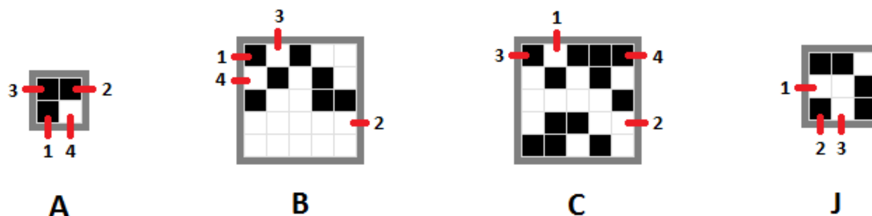
4.4 Krysninger

Før vi skal konstruere porter skal vi se på noen viktige mønstre som gjør det enklere å designe logiske porter. Gajardo konstruerte fire *krysninger* som vi kommer til å ha nytte av når vi skal konstruere våre porter [4]. Når to stier skal krysse hverandre, kaller vi det en krysning. En krysning er et mønster på rutenettet som hjelper mauren til å krysse en sti. Tidligere nevnte vi at vi må passe på stien til mauren, fordi den endrer seg hvis mauren har beveget seg på den. Dette medfører at vi ikke kan la to stier krysse hverandre ved å legge stien opp på hverandre. I figur 4.4 har vi en illustrasjon på idéen med en krysning. Vi kan ikke lage en krysning slik som på første bilde i figur 4.4.



Figur 4.4: Illustrasjon på hvorfor vi trenger krysninger.

Det er her de krysningene kommer til nytte. Krysninger er et mønster på rutenettet med innganger og utganger som har ulike funksjoner. De har funksjoner som gjør at mauren kan krysse en sti på ulike måter. De fire krysningene som Gajardo konstruerte kan vi se i figur 4.5.

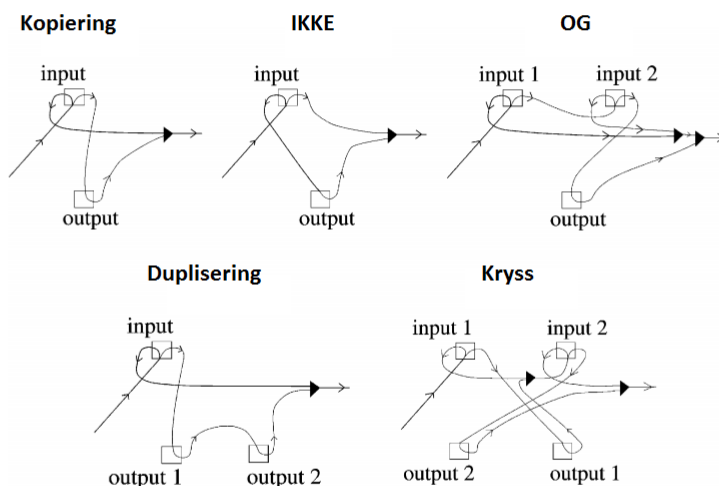


Figur 4.5: Gajardos sine krysninger.

- A brukes når mauren henter en hvit input, USANN. Mauren kommer skrått opp fra nedre venstre til høyre hjørne. I figur 4.5 A blir det at mauren kommer inn 1 og ut på 2. Senere når mauren kommer inn i 3, går den ut på 4.
- På B hvis mauren først kommer inn fra 1, går den ut på 2. Etter det hvis den går inn 3, kommer den ut på 4, men hvis mauren kommer inn først fra 3, kommer den ut fra 4 også.
- C er helt lik B bare med forskjellig tilstander og posisjoner.
- J brukes til å slå sammen to stier. Hvis mauren kommer inn fra 1 eller 2, kommer den ut fra 3.

4.5 Gajardos design av porter

Nå som vi har sett på de grunnleggende strukturene for logiske porter kan vi begynne å konstruere de logiske portene. Portene som Gajardo presenterte i artikkelen [4], designet hun slik:



Figur 4.6: Design av portene til Gajardo.

Den svarte trekanten i figuren 4.6 er symbolet som blir brukt for å slå sammen to stier til en sti (krysning J). Det var disse fem portene Gajardo brukte for å vise at Langtons maur kan utføre universelle beregninger. Med de to portene OG og IKKE Gajardo lagde, kan man lage alle de andre logiske portene. Gajardo konstruerte tre hjelpeporter som gjør at man kan utføre operasjoner fra boolske algebra. Disse tre portene er kopierings-, duplisering- og kryss-portene.

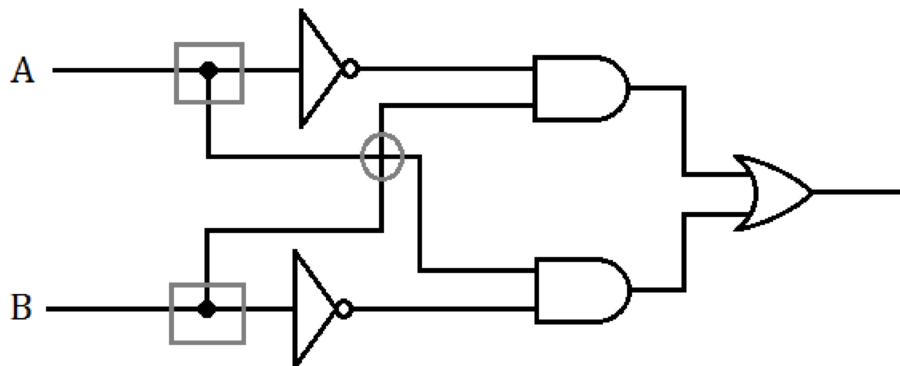
- Kopierings-port tar og kopierer en input og sender den videre nedover til outputen som brukes til å ta med en input videre til en port som er lenger nede. Eksempel på bruken av kopierings-porten er hvis mauren skal regne på $(A \vee B) \wedge C$, må mauren regne $(A \vee B)$ før det skal brukes en OG-port med input C. Da må vi bruke kopierings-porten på input C videre nedover slik at vi kan senere bruke den med OG-porten.

- Dupliserings-port tar og dupliserer en input. Dette skjer på nesten sammen måte som kopiering, bare at mauren går innom enda en

output. Når vi regner boolsk algebra med Langtons porter, kan en input brukes kun en gang. Dupliserings-porten brukes derfor for å duplisere en inputverdi slik at inputverdien kan brukes i flere porter.

- Kryss-porten er den mest kompliserte porten vi kommer til å se på i denne oppgaven. Den tar og bytter plass på to inputverdier slik at input 1 skrives ut i output 1 og input 2 skrives ut i output 2. Vi nevnte tidligere at mauren ikke kan krysse en sti. Kryss-porten brukes for å ta med en inputverdi gjennom en sti.

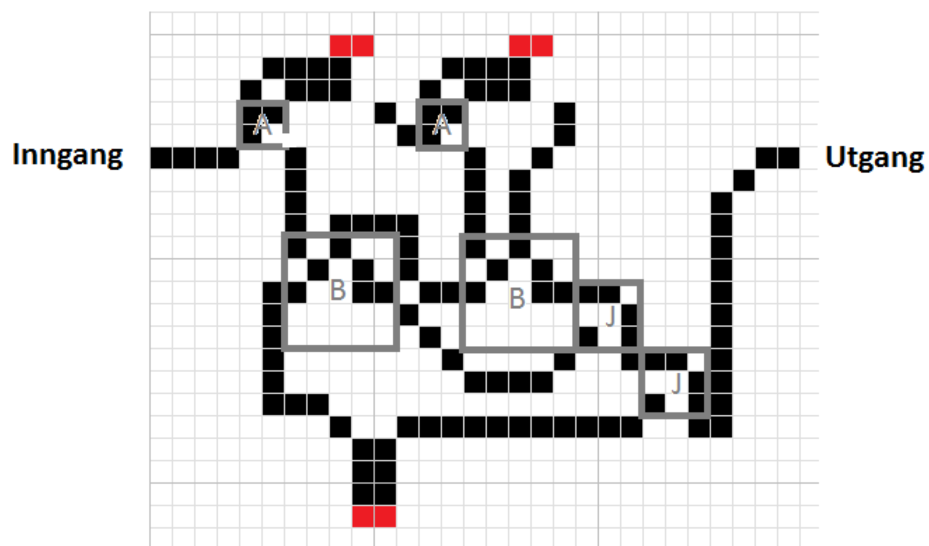
Eksempel 4.5.1. I figur 4.7 kan vi se eksempler på hvordan man bruker dupliseringsporten og kryssporten. I de punktene som er markert med grå firkant ser vi at ledningen splittes slik at A og B ledningen brukes i to porter. Det er der vi trenger dupliseringsporten slik at vi får to inputer til å bruke på to porter. Det punktet der det er markert med en grå sirkel brukes kryssporten for å bytte plass på to inputer. Vi skal senere se på flere eksempler på bruken av hjelpeporter etter vi har definert alt vi trenger.



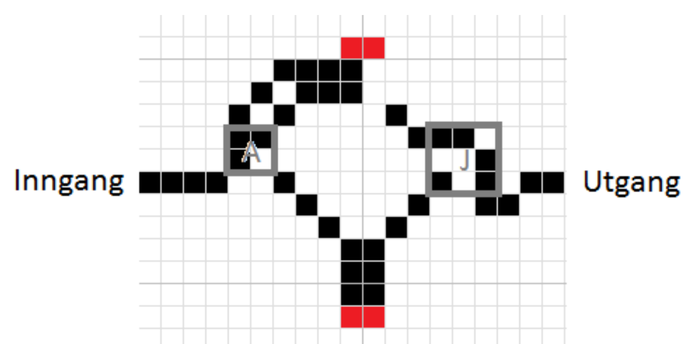
Figur 4.7: Eksempel hvor vi trenger å bruke hjelpeporter.

4.6 Gajardos porter

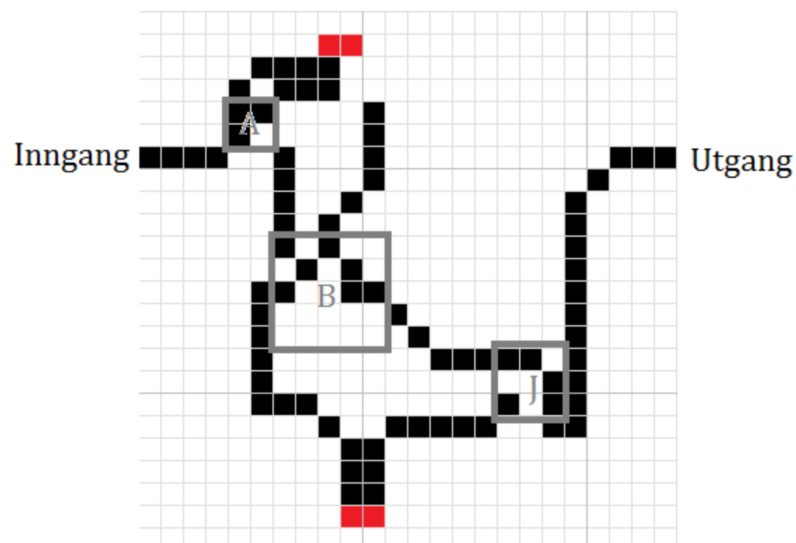
Ut fra figur 4.6 designet Gajardo disse portene. Alle portene hun konstruerte har inngang fra venstre mot høyre.



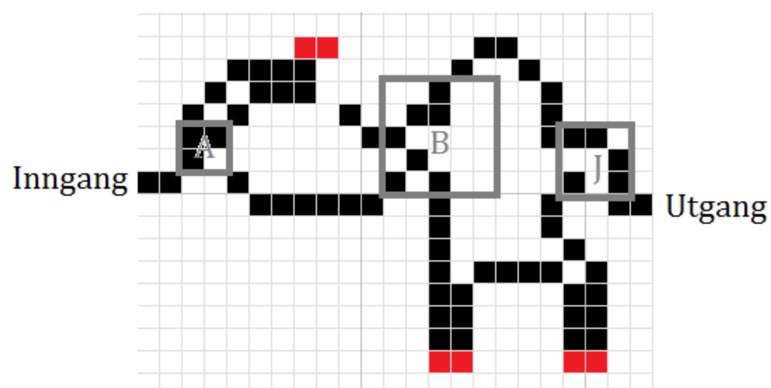
Figur 4.8: OG-port



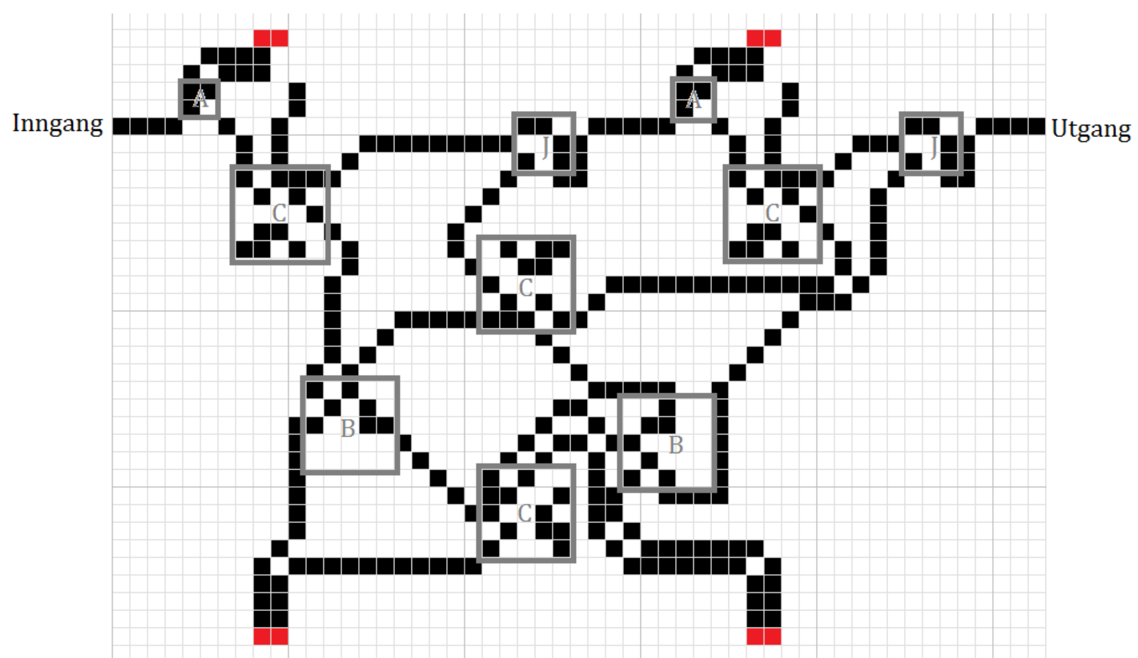
Figur 4.9: IKKE-port



Figur 4.10: Kopierings-port



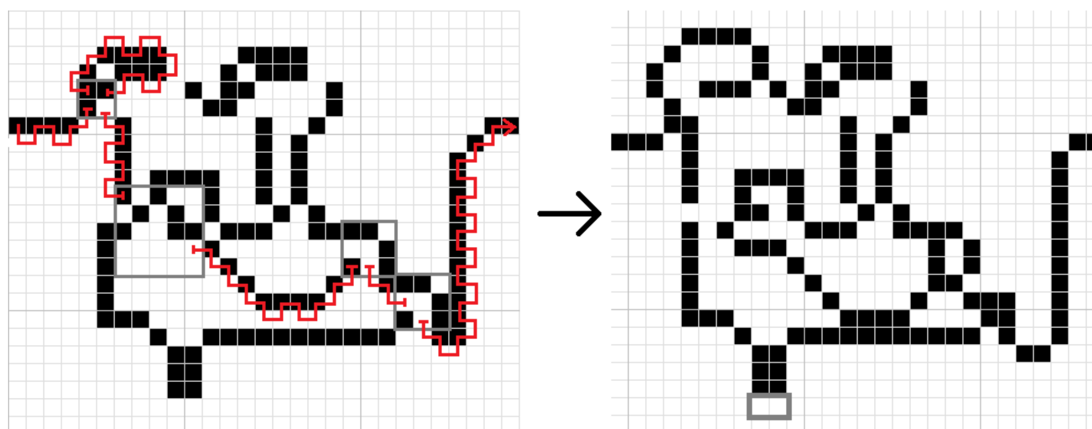
Figur 4.11: Dupliserings-port



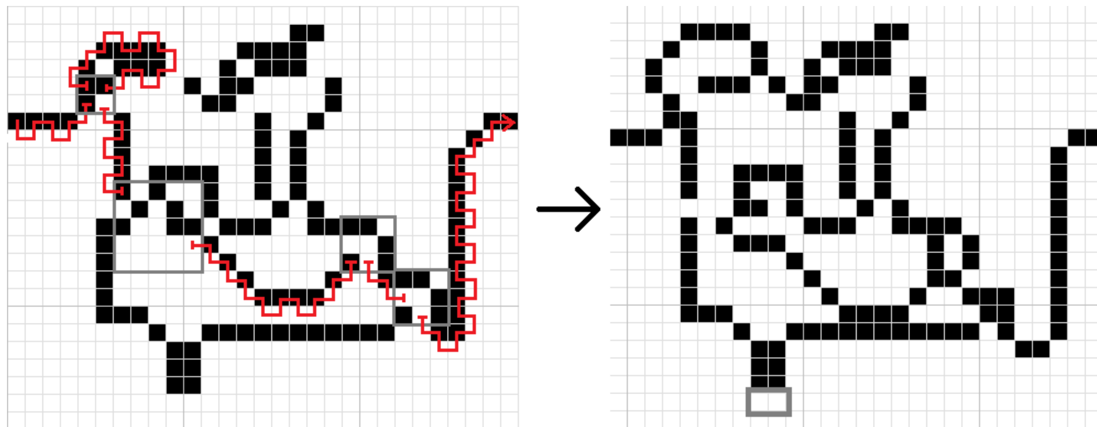
Figur 4.12: Kryss-port

4.7 Simulering på en port

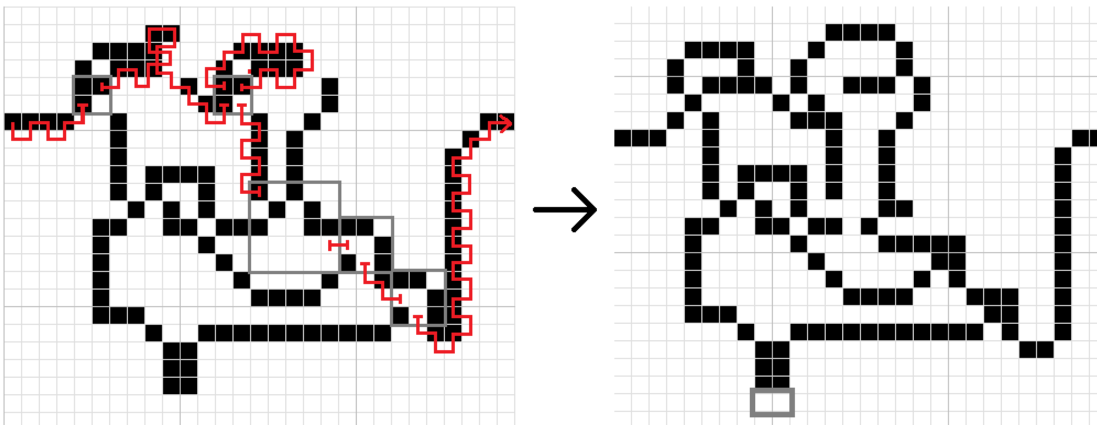
For å forstå hvordan portene med Langtons maur bedre, har vi valgt å vise en demonstrasjon med OG-porten. I figurene under har vi simulert mauren på OG-porten med de mulige inputene. I hver figur, viser bildet til venstre hvordan mauren beveger seg gjennom porten og bildet til høyre er hvordan porten ser ut etter at mauren har gått gjennom porten. Vi har markert med røde streker hvordan mauren beveger seg. Vi har ikke tatt med detaljer på hvordan mauren beveger seg gjennom hver krysning. Vi markerte hvor mauren gikk inn og ut av hver krysning. På bildet til høyre på hver figur har vi markert output cellene med en grå boks. Ut fra det vi kan observere, er ikke porten mulig å bruke lenger. Porten kan sees på som en gangs bruk, fordi mauren endrer farger på de cellene mauren går på og "ødelegger" porten. Portene har forskjellige slutt resultater fordi mauren beveger seg forskjellig ut fra hvilken input den blir simulert med. Det vil si mauren "ødelegger" forskjellige stier og krysninger for de forskjellige inputene. De som har nesten lik slutt resultat på bildet til høyre er figur 4.13 og 4.14 fordi på en OG-port, kan det oututes USANN så lenge den første inputen er USANN og mauren trenger ikke å se på input2.



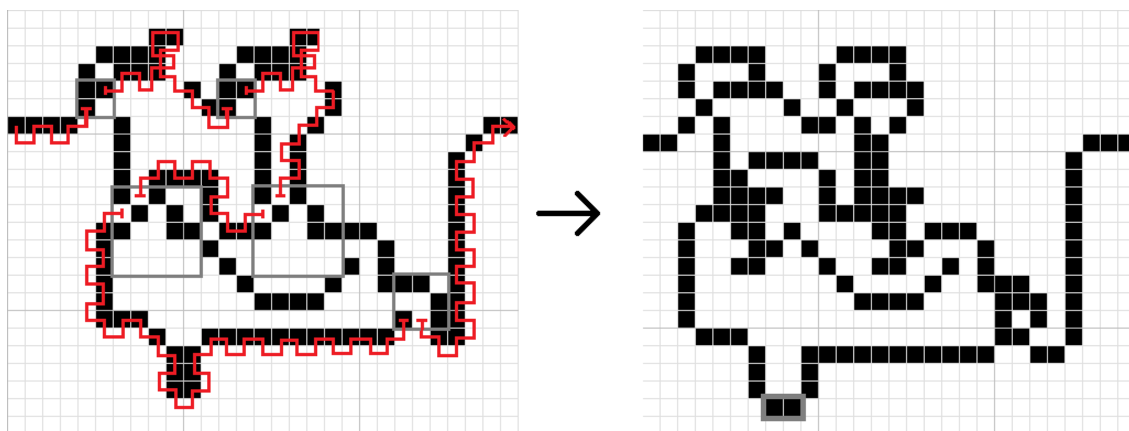
Figur 4.13: Simulering av OG-port med input1 = USANN og input2 = USANN.



Figur 4.14: Simulering av OG-port med input1 = USANN og input2 = SANN.



Figur 4.15: Simulering av OG-port med input1 = SANN og input2 = USANN.



Figur 4.16: Simulering av OG-port med input1 = SANN og input2 = SANN.

4.8 Sammendrag

I dette kapittelet har vi gått nøye gjennom det som har blitt beskrevet av Gajardo i [4]. Vi har sett på hvordan Gajardo definerte portene hun konstruerte og hvordan mauren ved hjelp av en sti leser inn inputer og outputer. Input og output er beskrevet som to hvite eller to svart celler ved siden av hverandre. I figurene er inputer og outputer markert som røde felter. Det viktigste av alt er de fire krysningene. En logisk port med Langtons maur består egentlig av bare stier og krysninger som vi har pakket sammen og kalt det for en port. Hjelpeportene er viktig for å kunne regne på boolske regneoperasjoner som vi skal se på i de neste kapitlene. Hjelpeportene brukes til å flytte på inputer og outputer til de cellene i rutenettet vi vil ha dem, slik at vi kan senere bruke verdiene til å regne på de eller ha dem som resultatverdier. Alt vi har sett på i dette kapittelet skal vi bygge videre på i de neste kapitlene.

Kapittel 5

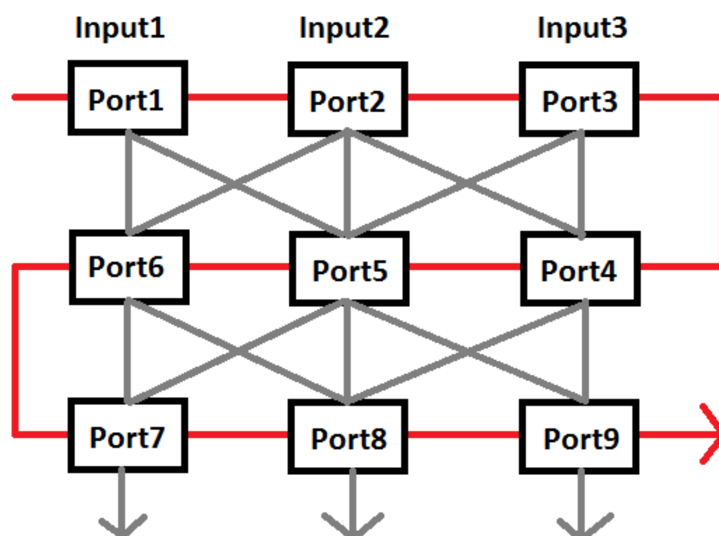
Beregninger av porter og kretser med Langtons maur

I dette kapitlet skal vi bruke det vi har definert i de tidligere kapitlene til å designe modeller for å lage nye porter og kretser. Før vi skal konstruere portene og sette dem sammen til kretser skal vi se på hvordan vi kan koble flere porter sammen. Vi kommer også til å støte på problemer under konstruksjonen av blant annen XOR-porten.

5.1 Beregninger

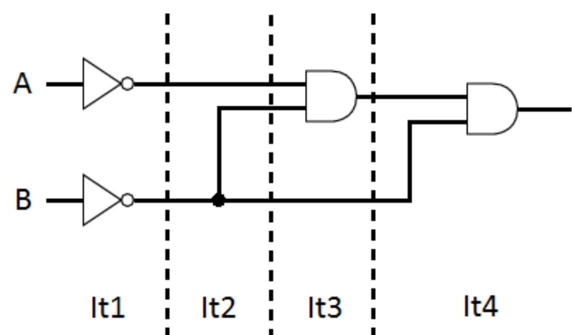
I denne seksjonen skal vi se hvordan man bruker portene til å utføre operasjoner i boolsk algebra. Ved å koble sammen innganger og utganger kan mauren utføre en *beregning* ved å gå gjennom alle portene. Outputen vil vises nederst under hver av portene slik at de neste portene i de neste *iterasjonene* kan bruke outputen. En beregning i denne sammenhengen er når mauren har gått gjennom alle portene som er koblet sammen. Beregningen gjøres i iterasjoner der hver rad blir kalt en iterasjon. I elektronikk pleier man å sette opp portene slik at ledningen går fra venstre mot høyre, men for Langtons maur har vi valgt å sette opp portene under hverandre. Måten vi skal utføre en beregning på er vist i figur 5.1.

I figur 5.1 starter mauren på venstre side og går til høyre. Vi ser også på den andre iterasjonen at inngangen på hver av portene (port 4, 5 og 6) er på høyre side og utgangen på venstre siden. Vi skal i den neste seksjonen vise hvordan vi skal konstruere de portene som går fra høyre og ut på venstreside. Det er ikke like lett som å speile om en port. I figur 5.2 ser vi



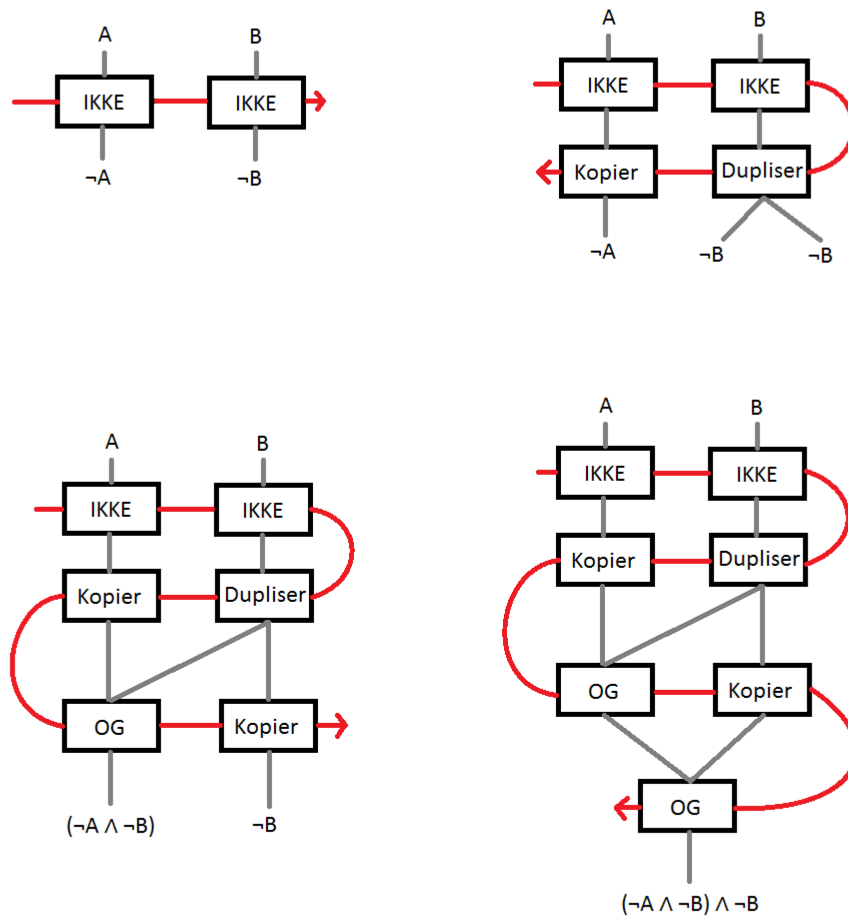
Figur 5.1: Idéen om hvordan vi skal koble sammen kretser med Langtons maur porter. Den røde streken representerer mauren og de grå strekene representerer input- og outputverdier.

hvordan vi deler opp i iterasjoner for beregningen med Langtons maur.



Figur 5.2: Eksempel på oppdeling av iterasjoner i en krets.

Eksempel 5.1.1. Vi skal konvertere kretsen i figur 5.2 til vår Langtons maur krets. I figuren ser vi at mauren bruker fire iterasjoner for å regne ut kretsen. I figur 5.3 ser vi hvordan vi bygger opp kretsen.



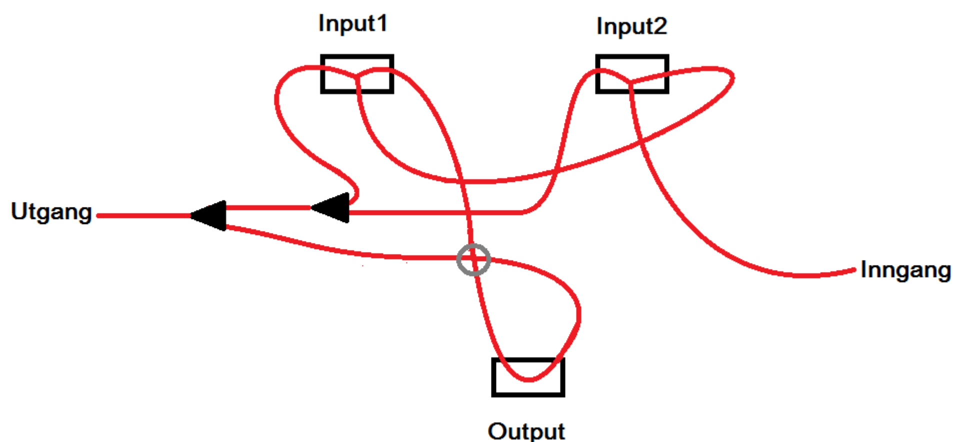
Figur 5.3: Illustrasjon på hvordan vi skal lage kretsen i figur 5.2 steg for steg.

5.2 Porter fra høyre mot venstre

I figur 5.1 ser vi at port nummer 4, 5 og 6 er porter som har inngang på høyre side og utgang på venstre side. I denne seksjonen skal vi se på hvordan vi kan konstruere slike porter. I de to første metodene vi skal skissere, bruker vi svart tilstand som stien hvor rutenettet har hvit som initialtilstand. Dette er ikke en fullstendig beskrivelse. I den tredje metoden skal vi bruke en metode hvor vi initialiserer brettet svart og lar stien være hvit. I [4] gis det ingen detaljer om hvordan porter med inngang på høyre side og utgang på venstre side faktisk skal konstrueres.

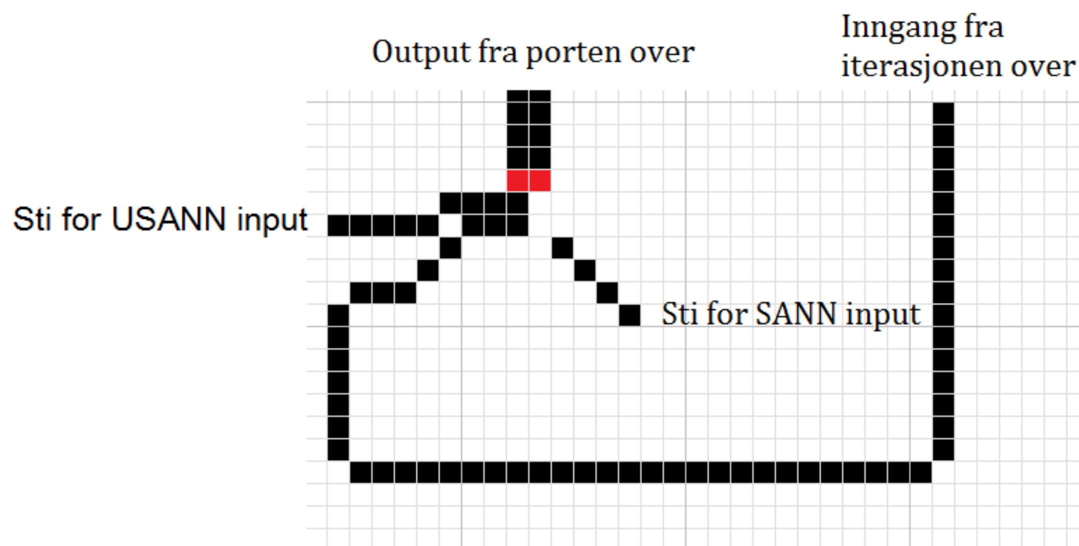
5.2.1 Metode 1

Den naive metoden er å lage en sti rett ned til den neste iterasjonen. Da kommer vi til å støte på det problemet med at mauren kommer fra høyre side på oversiden av stien. Dette medfører at vi må ta en u-sving for å lese en input. U-svingen gjør at mauren må krysse stien for SANN input (som er output fra porten over). Det medfører at når vi konstruerer en port, må vi også konstruere en port for motsatt vei og de vil da se forskjellige ut. I figur 5.4 har vi et eksempel med en OG-port.



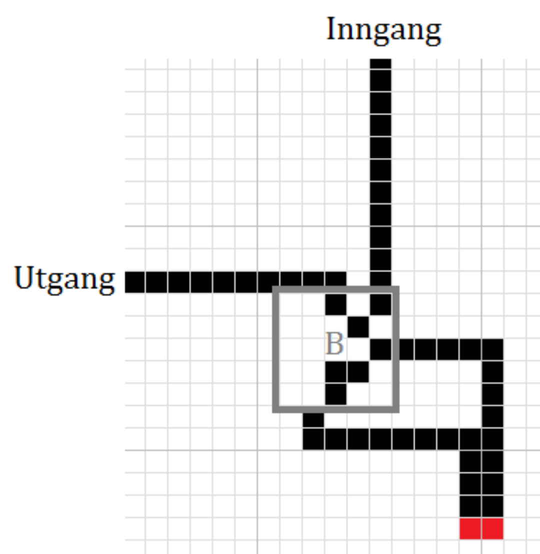
Figur 5.4: Design av OG-port med inngang på høyreside og utgang fra venstreside.

Når det gjelder utskriving av output, må mauren ta seg en u-sving slik at den krysser sin egen sti. Fordi når mauren skriver ut outputen beveger den langs "output søylen" fra venstre mot høyre. En måte å løse dette på er å for eksempel bruke en krysning B, som er snudd 90 grader med



Figur 5.5: Metode 1

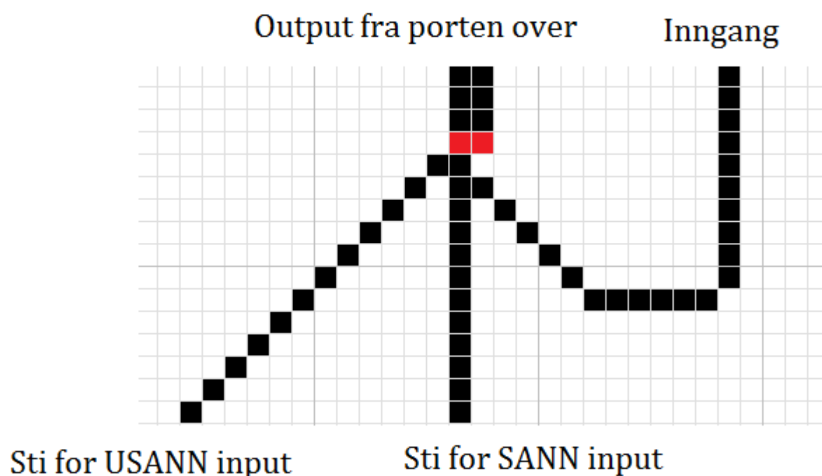
klokka, til å krysse stien på vei til utgangen på venstre side slik som i figur 5.6.



Figur 5.6: Skrive ut output med metode 1.

5.2.2 Metode 2

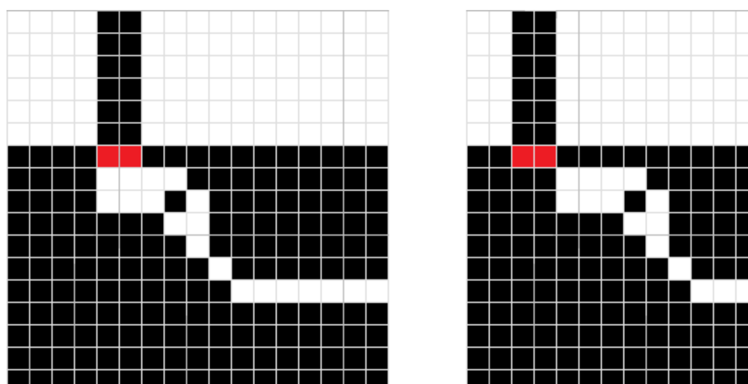
Den andre metoden er å lese inn inputen på en ny måte slik at man slipper å krysse stien. Da vil det bli en krysning mindre for hver port. I figur 5.7 kan vi se en av måtene å konstruere dette på. For output kan man her også bruke metode 1.



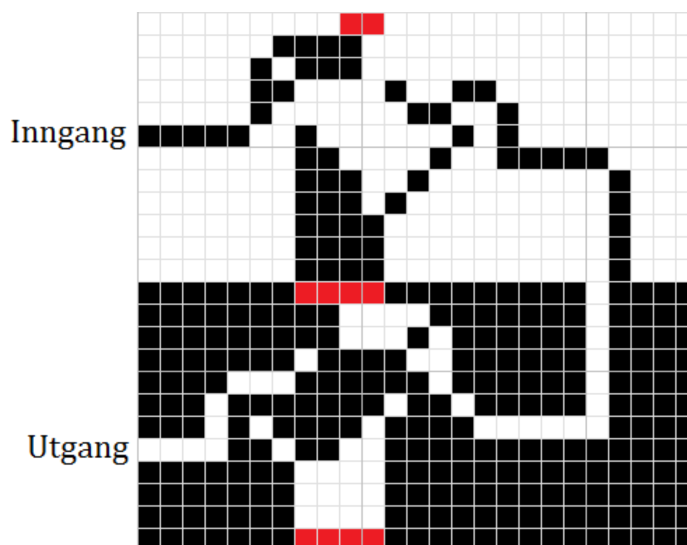
Figur 5.7: Metode 2.

5.2.3 Metode 3

Den tredje og siste metoden vi skal nevne i denne oppgaven er en metode som bytter farge på stien. I figur 5.9 er det et eksempel som viser hvordan man kan gjøre dette med to IKKE porter under hverandre. Vi ser i den første iterasjonen at stien er svart med hvit bakgrunn og på den andre iterasjonen, har vi svart bakgrunn med hvit sti. Dette gjør at vi kan speile om alle krysninger og stier vi har sett på. Dette medfører at hele iterasjonene får hvit og svart bakgrunn alternerende. Når vi sier bakgrunn i denne sammenhengen, mener vi bakgrunnsfargen for en port. Vi skal se nærmere på dette i de neste seksjonene. Når det kommer til outputen ser vi i figur 5.9 at input og output feltene mellom iterasjonen er fire celler istedenfor to celler. Når vi bytter bakgrunnsfarge for en port blir output og input forskyvet slik at de ikke kan kobles riktig, men kan kobles på en av de to måtene som vises i figur 5.8. Vi endrer derfor outputen til fire celler. Output feltene til portene med svart bakgrunn må endres til å ha standard tilstand SANN (svart) for å bevare "speilvendt" egenskapet.



Figur 5.8: Innlesing av en output "søyle" med bredde to fra forrige iterasjon.



Figur 5.9: Metode 3.

Selv om vi kan speile en port, er det ikke sikkert at det betyr at vi beholder egenskapen til porten. Med egenskap i denne sammenhengen mener vi outputen som skrives ut. Grunnen er at vi vil at svart tilstand skal fortsette å representere SANN og hvit tilstand representere USANN. Hvis vi hadde byttet svart tilstand til USANN og hvit tilstand til SANN hadde vi kunne ha speilet alle porter og porten hadde beholdt egenskapen. Av de portene som vi har sett på som beholder egenskapen når vi speiler om porten er: IKKE, kopierings, dupliserings og kryssporten. Vi skal senere se hvordan porter fungerer speilvendt.

5.3 Mulige porter

Før vi skal se på hvordan vi skal konstruere en XOR-port skal vi se på hva slags porter vi kan lage med Langtons maur. I artikkel [4] konstruerte Gajardo to logiske porter OG og IKKE som hun brukte til å lage alle de andre portene. I denne seksjonen skal vi se på hvilken mulige porter vi kan konstruere direkte. Det vil si at vi konstruerer porten med kun en iterasjon. Vi vet at hvis mauren leser en input har den tre alternativer for en sannhetsverdi:

- Gå ut til utgangen av porten og la outputen være USANN (hvit tilstand).
- Gå og lese neste input.
- Gå ned til output, bytte outputen til SANN og gå videre ut til utgangen.

Hver input er enten sann eller usann. For hver av disse verdiene, velges nøyaktig ett av de tre alternativene over. Det vil si mauren har seks mulige stier som kan konstrueres når mauren har lest en inputverdi. Bortsett fra den siste inputen som blir lest, den kan ikke lese neste input. Vi skal se på tre av de mest interessante mulighetene som vi kan få (vi antar at inputen blir lest av mauren):

- Hvis mauren går ut til utgangen etter den har lest inputen for begge sannhetsverdiene, vil vi få en port som er kontradiktorisk. Fordi uansett hva mauren leser, går mauren ut og lar outputen være USANN.
- Det motsatte tilfellet er, hvis for begge sannhetsverdiene så går mauren ned til outputen og skriver ut SANN. Da vil porten være en tautologi fordi den outputer ut SANN uansett. Det vil si at porten skriver ut SANN uansett hvilket input den får.
- Det neste tilfelle er når mauren skal lese det neste input for begge sannhetsverdiene. Det er det samme som å ikke lese den første inputen. Fordi for uansett sannhetsverdi kommer mauren til å se på den neste inputen.

I figur 5.10 ser vi forskjellige sannhetsverditabeller for to variabler. De tabellene er de som kan konstrures direkte med en port med to inputer.

Det vil si at de ikke trenger flere porter eller iterasjoner for å uttrykke de. Vi ser her at XOR-tabellen og XNOR ikke er med i figuren. I seksjon 5.4 skal vi forklare hvorfor det er ingen åpenbar og enkel måte å konstruere XOR-port og XNOR på.

A	B	OG
1	1	1
1	0	0
0	1	0
0	0	0

A	B	ELLER
1	1	1
1	0	1
0	1	1
0	0	0

A	B	3
1	1	1
1	0	0
0	1	1
0	0	1

A	B	4
1	1	0
1	0	0
0	1	1
0	0	0

A	B	5
1	1	1
1	0	1
0	1	0
0	0	1

A	B	NAND
1	1	0
1	0	1
0	1	1
0	0	1

A	B	7
1	1	0
1	0	1
0	1	0
0	0	0

A	B	NOR
1	1	0
1	0	0
0	1	0
0	0	1

Figur 5.10: Sannhetsverditabell til de mulige portene vi kan konstruere.

I figur 5.11 er fire andre porter som kan konstrueres. Disse sannhetsverditabellene kan forkortes til sannhetsverdi tabeller med en variabel.

A	B	
1	1	1
1	0	1
0	1	0
0	0	0

A	B	
1	1	1
1	0	0
0	1	1
0	0	0

A	B	
1	1	0
1	0	0
0	1	1
0	0	1

A	B	
1	1	0
1	0	1
0	1	0
0	0	1

Figur 5.11: Første tabell kan forkortes til A, andre til B, tredje til $\neg A$ og siste til $\neg B$

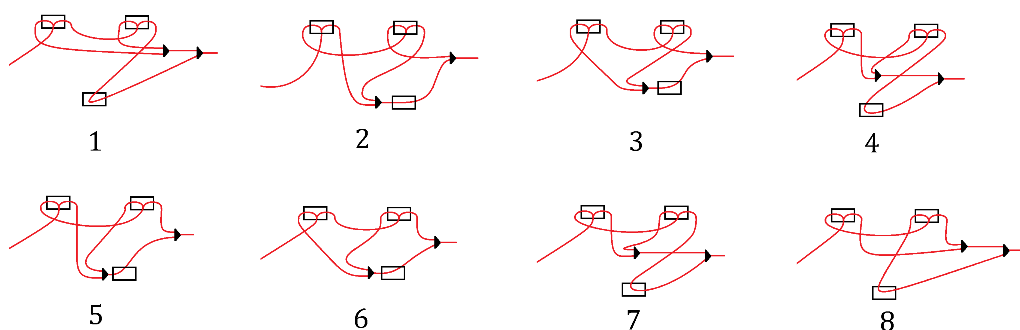
A	B	XOR
1	1	0
1	0	1
0	1	1
0	0	0

A	B	XNOR
1	1	1
1	0	0
0	1	0
0	0	1

Figur 5.12: Sannhetsverditabell for XOR og XNOR.

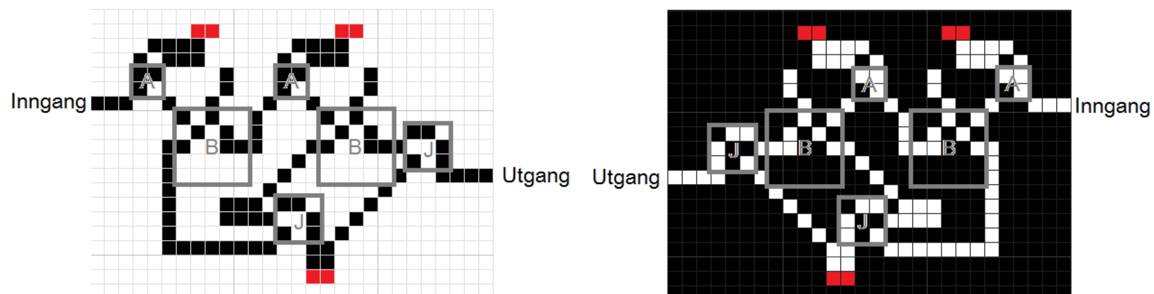
5.3.1 Porter

Nå som vi har sett på hvilken porter som er mulig å lage skal vi nå i denne seksjonen demonstrere hvordan vi kan lage porter med to inputer. Tidligere nevnte vi også at når vi speiler om en port ved bruk av metode 3 hender det at en port blir en annen port speilvendt. I figur 5.13 har vi designet hvordan de åtte portene i figur 5.10 skal se ut.

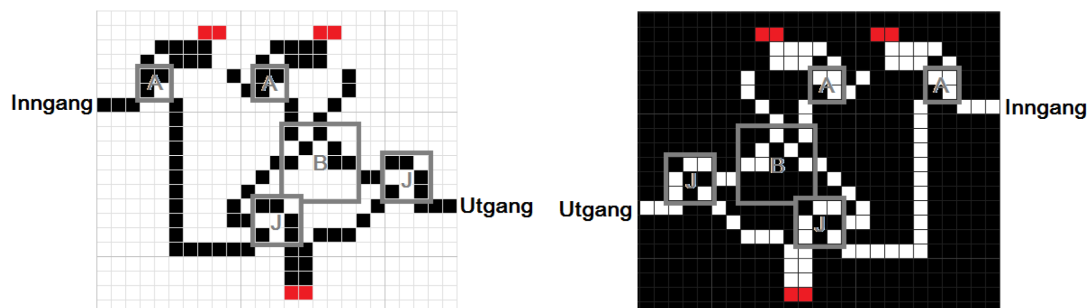


Figur 5.13: Design av de mulige portene vi kan konstruere.

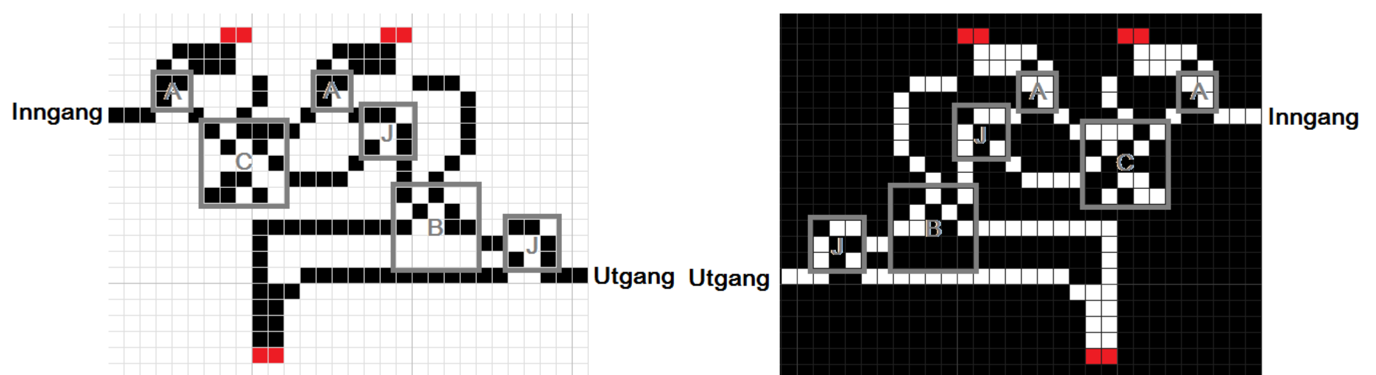
For speilvendte porter eller porter som går inn fra høyre og ut fra venstre har vi laget en tabell som forteller hvilken port som tilsvarer den speilvendte porten. I den første raden ser vi at hvis vi speiler om OG-port ved bruk av metode 3 tilsvarer det en ELLER-port.



Figur 5.14: ELLER-port og OG-port



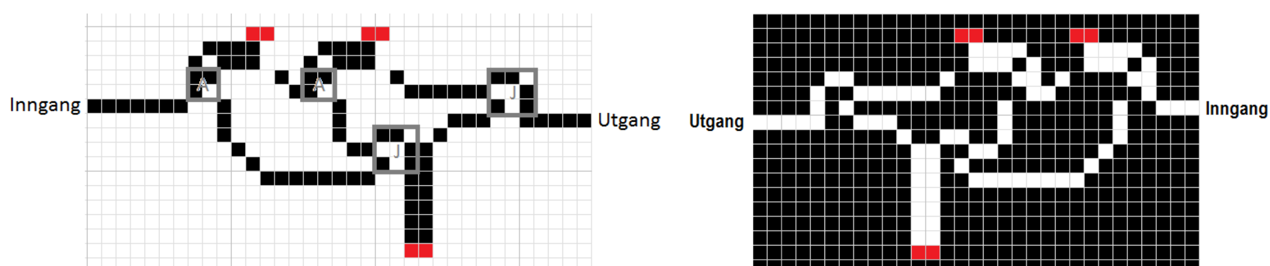
Figur 5.15: Port nr. 3 og port nr. 4. Port nr. 3 er også kjent som implikasjon fra utsagnslogikk.



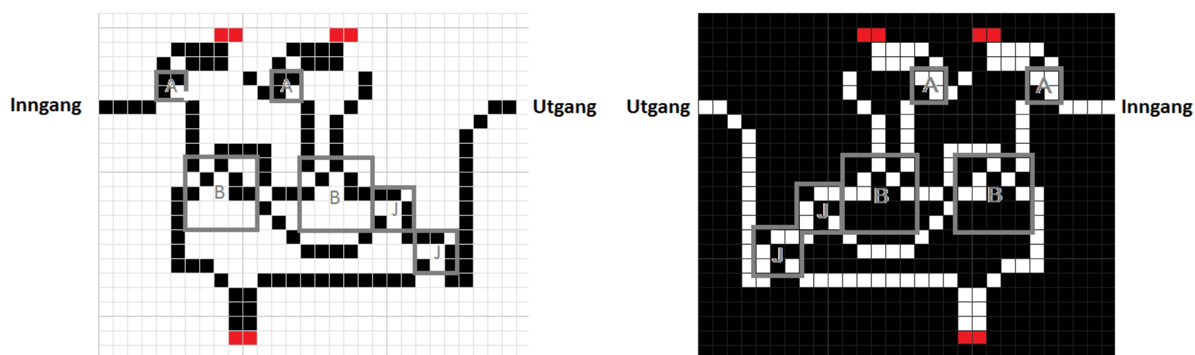
Figur 5.16: Port nr. 4

Venstre høyre	Høyre venstre (speilvendt)
OG	ELLER
ELLER	OG
Port nr. 3	Port nr. 4
Port nr. 4	Port nr. 3
Port nr. 5	Port nr. 7
NAND	NOR
Port nr. 7	Port nr. 5
NOR	NAND

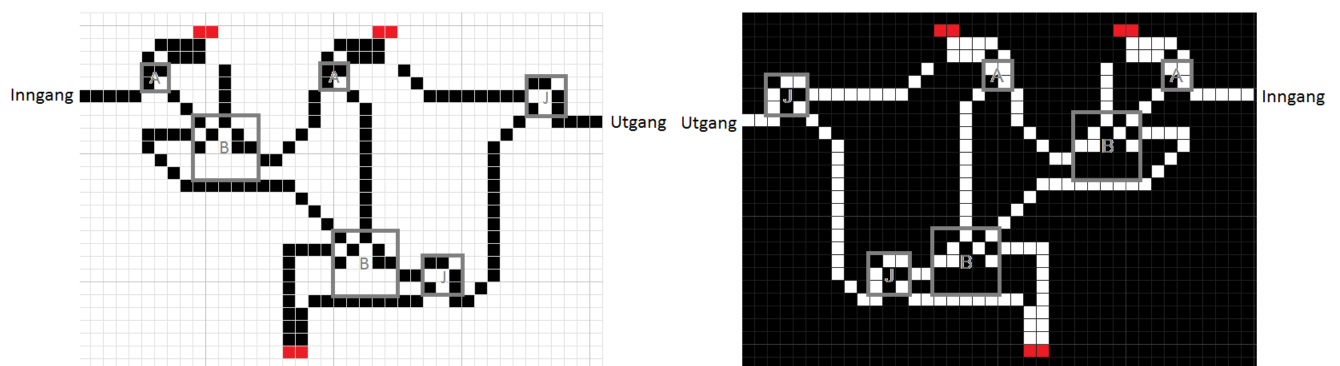
Tabell 5.1: Oversikt over porter når de speiles.



Figur 5.17: NAND-port og NOR-port



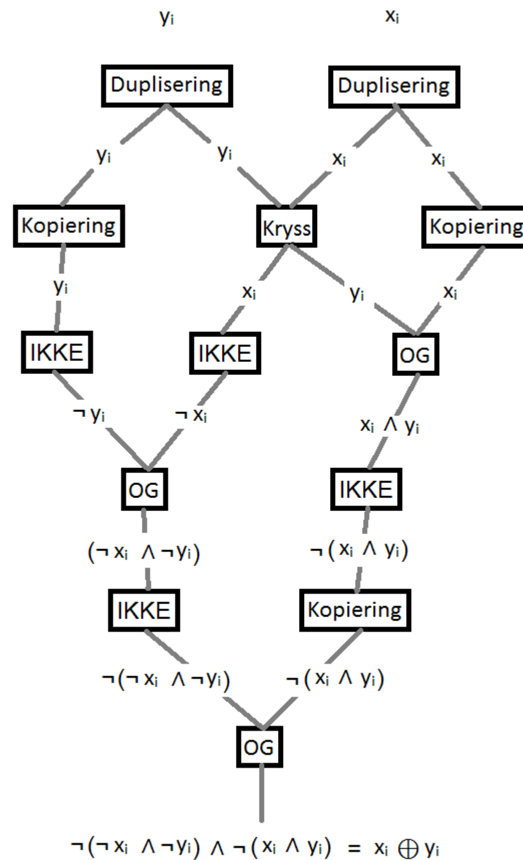
Figur 5.18: OG-port og ELLER-port



Figur 5.19: NOR-port og NAND-port

5.4 XOR-port

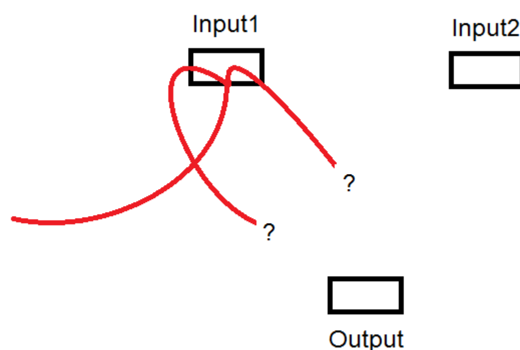
I forrige seksjon nevnte vi at XOR-porten er en av de portene som vi ikke var så enkel å konstruere direkte med krysningene Gajardo konstruerte, men må bruke flere porter og iterasjoner for å uttrykke det. Nedenfor i figur 5.20 ser vi et eksempel på hvordan vi kan representere en XOR-port med portene som Gajardo konstruerte. Konstruksjonen bruker totalt 13 porter og 6 iterasjoner.



Figur 5.20: Design av hvordan vi kan lage XOR-port med portene Gajardo konstruerte.

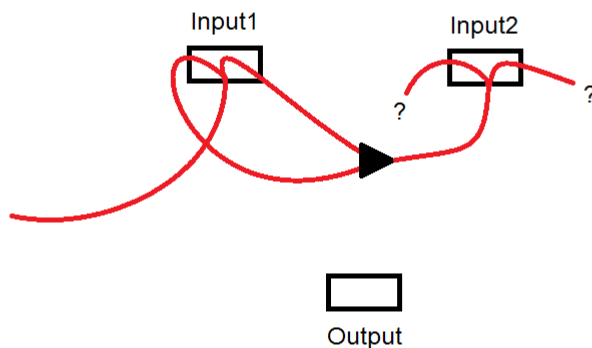
La oss tenke at vi skal lage en XOR-port slik at vi trenger kun en port og en iterasjon. Vi skal vise at det blir vanskelig å lage en slik port med de krysningene vi har. Vi vet at mauren ikke har noe form for hukommelse og kan ikke huske verdier den har lest på inputen. Mauren følger bare stien helt til den kommer til utgangen. For hver input mauren leser, kan

den enten gå til høyre eller venstre for inputen, avhengig av hva inputen er. Dette fører til det tredje punktet fra seksjon 5.3 der det tilfellet hvor mauren sjekker neste input for begge inputverdier.



Figur 5.21: XOR med Gajardos krysning.

Mauren kan ikke outpute SANN før den har sett begge inputene. Dermed ender det at mauren leser input2 uansett om input1 er SANN eller USANN.

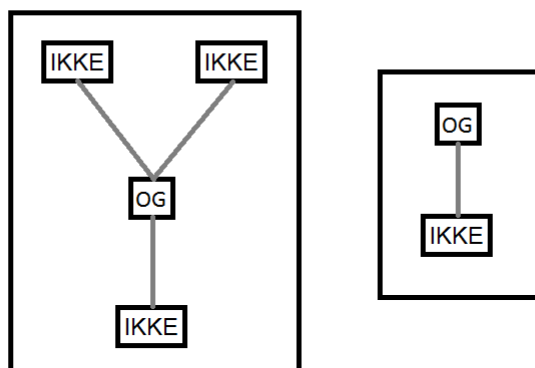


Figur 5.22: XOR med Gajardos krysning.

Problemet er når mauren har lest input2 så vet vi ikke hvordan vi skal fortsette å konstruere stien for SANN og USANN. Med krysningene Gajardo konstruerte vil det være vanskelig å lage direkte en XOR-port med Langtons maur.

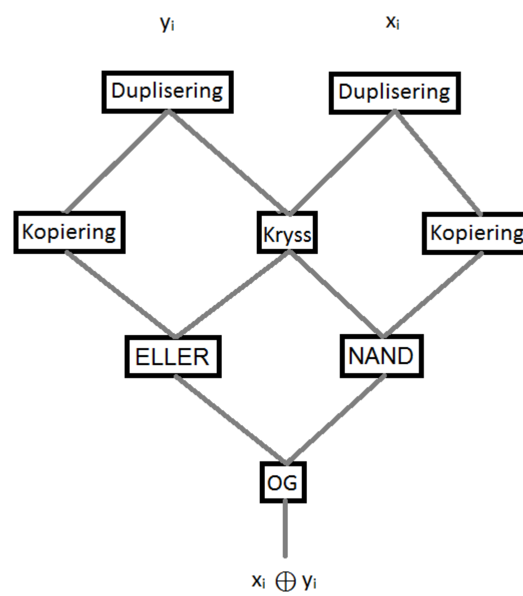
5.5 Forkortning av XOR-porten

Selv om det er vanskelig å lage XOR-porten med Gajardos sine krysninger, finnes det andre måter å forkorte XOR-porten vi konstruerte tidligere i figur 5.20. Hvis vi titter nærmere på den tidligere XOR-porten vi konstruerte i figur 5.20, kan vi se at det er et par forenklinger som gjør at mauren slipper å gå gjennom så mange porter og iterasjoner.



Figur 5.23: Forkorte fra venstre bildet til høyre bildet.

Hvis vi ser på disse to beregningene i figur 5.23 som er en delberegning i den tidligere XOR-porten, ser vi at den beregning til venstre tilsvarer en ELLER-port og den til høyre tilsvarer en NAND-port. ELLER-port og NAND-port kan vi lett konstruere ved å bruke krysningene til Gajardo. Med ELLER- og NAND-port blir det mindre porter og iterasjoner mauren må gjennom. I figur 5.24 kan vi se en forkortet versjon av XOR-porten.

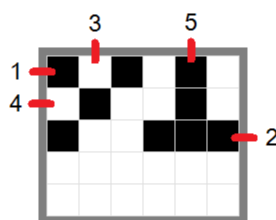


Figur 5.24: Forkortet representasjon av XOR-port

5.6 Krysning X

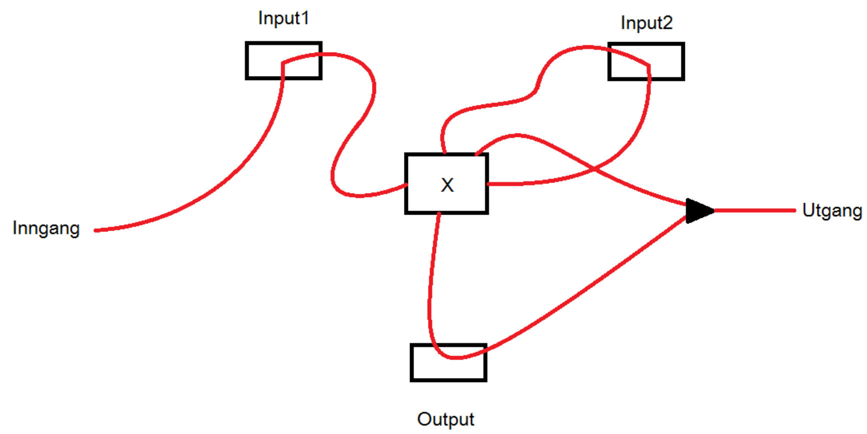
I denne seksjonen skal vi presentere en løsning på XOR-porten. Ved å kun bruke krysningene Gajardo konstruerte, klarte vi ikke å konstruere XOR-porten. Vi skal se på en krysning vi har konstruert, krysning X. Dette er en krysning med tre utganger og to innganger. Idéen med denne er å bruke krysningen som en form for hukommelse for mauren. Beskrivelsen til krysning X ser slik ut:

- Hvis mauren går inn 1, går den ut 2.
- Hvis mauren går inn 3, går den ut 5.
- Hvis mauren går 1 og ut 2 og senere går inn 3, går mauren ut fra 4.

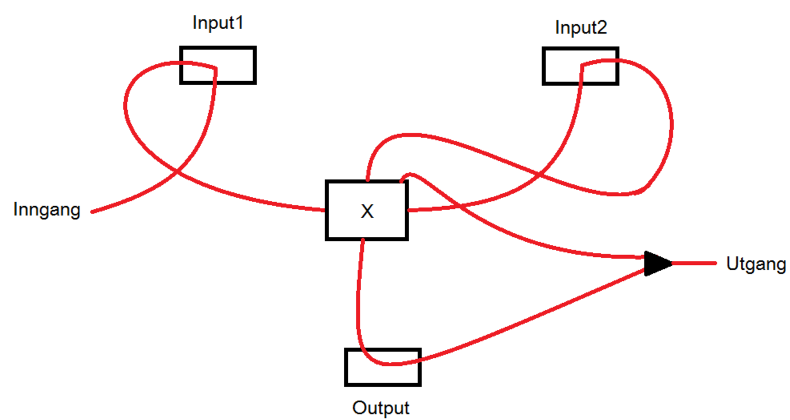


Figur 5.25: Krysning X

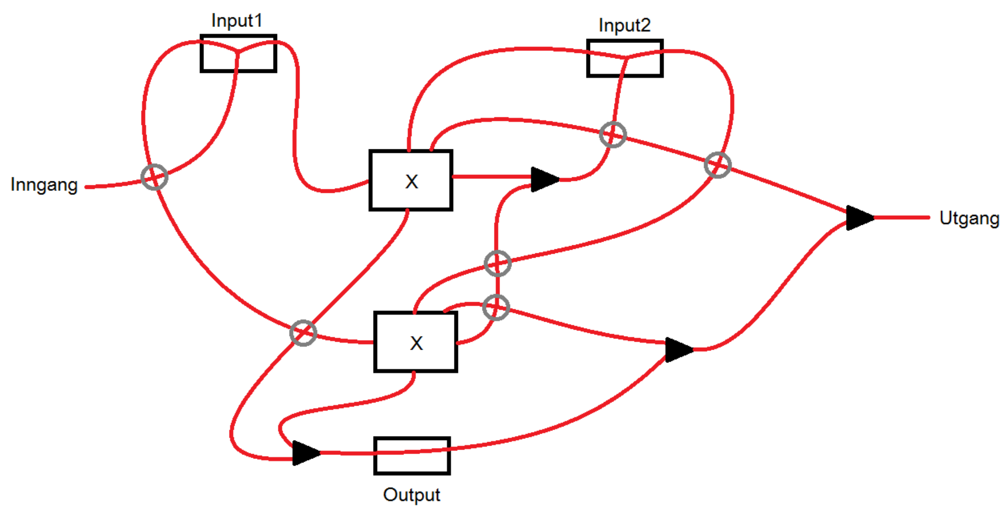
Måten vi får mauren til å "huske" en verdi på er å bruke inngang 1. Idéen er at mauren skal inn inngang 3 uansett og ut fra enten utgang 4 eller 5. Inngang 1 skal være den inngangen som bestemmer utfallet av utgangen til inngang 3. På denne måten kan mauren huske en verdi. Dette kan utnyttes til å lage en XOR-port. I figur 5.26 har vi illustrert hvordan vi kan bruke krysning X til å konstruere en XOR-port. For å gjøre det enklere har vi ikke tegnet krysning A, B og C i figuren. I figur 5.26 har vi håndtert det tilfelle for input1 = SANN og input2 = USANN. I figur 5.27 har vi håndtert tilfellet for input1 = USANN og input2 = SANN. Vi designet figurene slik for å få bedre oversikt. Det steget som gjenstår er å slå sammen de to figurene sammen og koble stier riktig mot hverandre. I figur 5.28 ser vi hvordan man kan slå sammen 5.26 og 5.27. Vi må huske å slå sammen stiene til utgang 2 for begge krysning X fordi de skal sjekke samme input. Der det er markert med en grå ring skal det settes inn enten krysning A, B eller C. Resultatet til XOR-porten kan vi se i figur 5.29. Vi ser at XOR-porten er den største porten hittil med hensyn på antall krysninger.



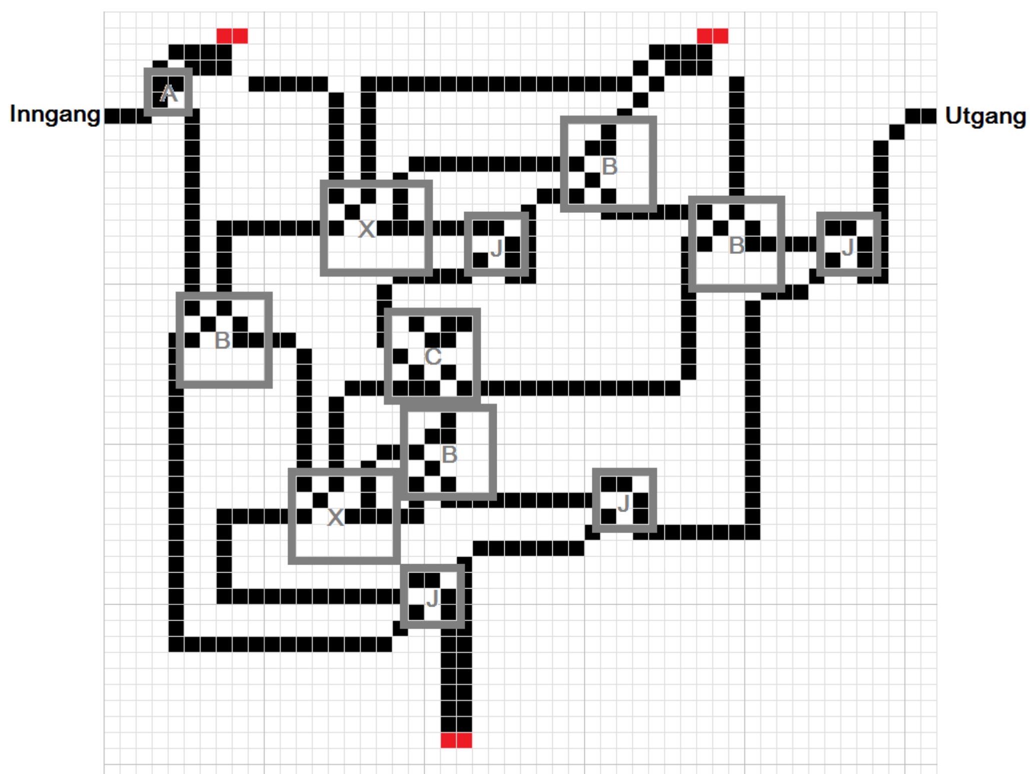
Figur 5.26: Design 1.



Figur 5.27: Design 2.



Figur 5.28: Design 3.



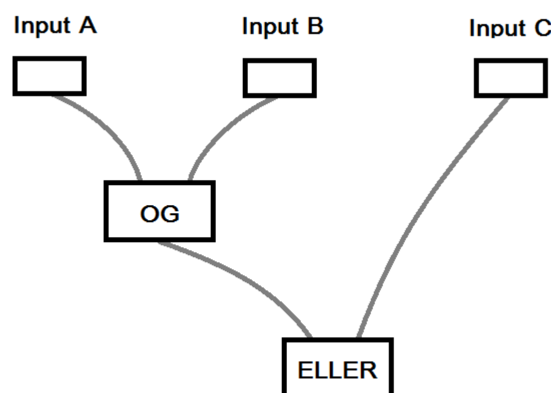
Figur 5.29: XOR-port

5.7 Eksempler på kretser og beregninger

I denne seksjonen skal vi se på bruken av Langtons maur portene. Husk at siden vi bruker metode 3 blir bakgrunnen til iterasjonene på beregningen alternerende svart og hvit. Vi må også huske å bruke tabell 5.1 for å lage porter med inngang fra høyre og utgang på venstre. For at portene skal passe inn når det skal kobles hender det at noen porter må strekkes slik at porten ikke er garantert å se lik som portene i seksjon 5.3.1.

5.7.1 Kopierings-port

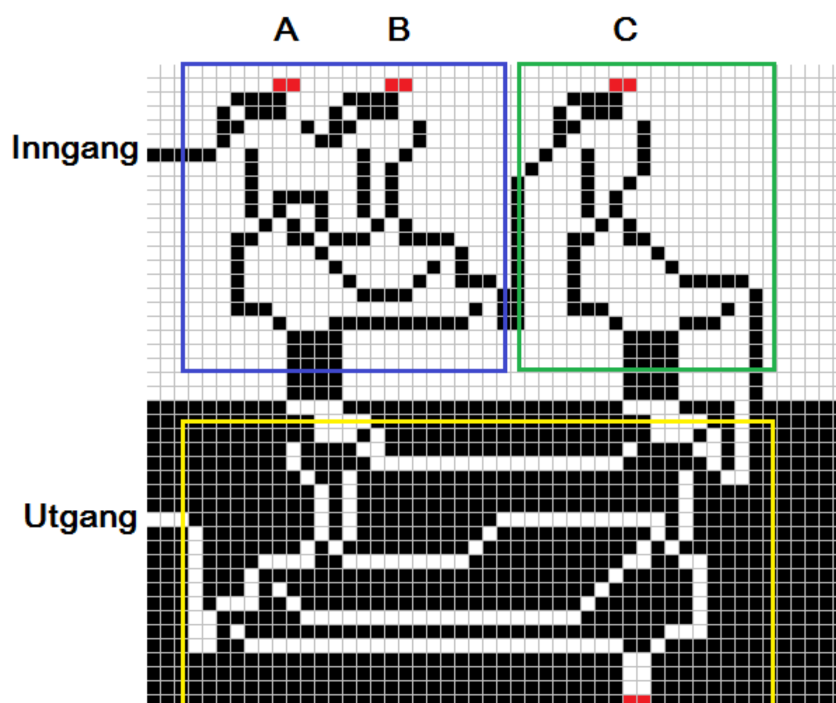
Vi skal i denne delseksjonen se på bruken av kopierings-port ved å konstruere modellen i figur 5.30. I figur 5.31 har vi konstruert en Langtons maur krets av modellen i figur 5.30. I figur 5.31 har vi tre input cellepar på toppen og en output cellepar på bunnen markert i rødt. I figuren ser vi OG-porten markert i en blå firkant, kopierings-porten er markert i grønt og ELLER-porten er markert i gult. Vi ser at OG-porten tar inn to inputverdier A og B. For å kunne bruke outputverdien til OG-porten sammen med C, må vi kopiere C slik at den kommer på samme iterasjon med outputverdien til OG-porten.



Figur 5.30: Demonstrasjon av kopierings-port.

5.7.2 Dupliserings-port

Tidligere nevnte vi at en inputverdi kan brukes kun en gang slik at hvis vi skal bruke en inputverdi flere ganger må vi duplisere inputverdien. I

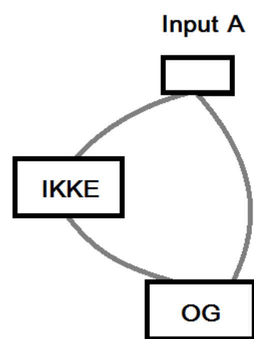


Figur 5.31: Demonstrasjon av kopierings-port.

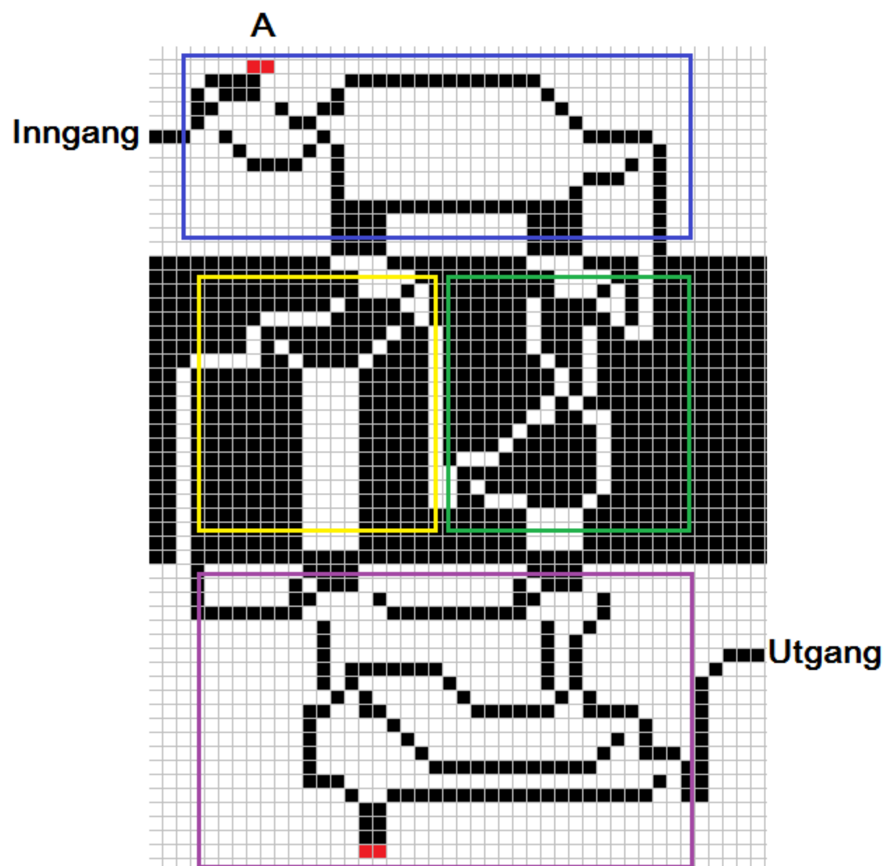
figur 5.32 har vi designet en modell hvor input A brukes i IKKE-porten og OG-porten. I figur 5.33 har vi laget en modell av figur 5.32. Den blå firkanten er en dupliserings-port, gul firkant er IKKE-port, grønn firkant er kopierings-port og lilla firkant er en OG-port. Vi ser at dupliserings-porten sender ut to outputer som brukes i IKKE-porten og kopierings-porten.

5.7.3 Kryss-port

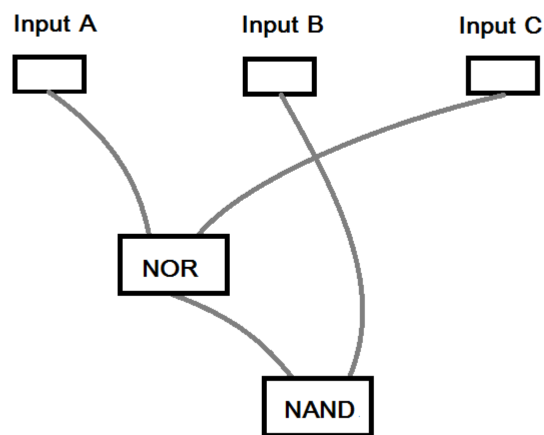
En kryss-port brukes for å krysse en sti med en inputverdi. I figur 5.34 ser vi at input C skal krysse gjennom stien til input B sin sti. I figur 5.35 er den blå firkant en kryss-port, gul og oransje firkanter kopierings-porter, rosa firkant en NOR-port og grønn firkant en NAND-port.



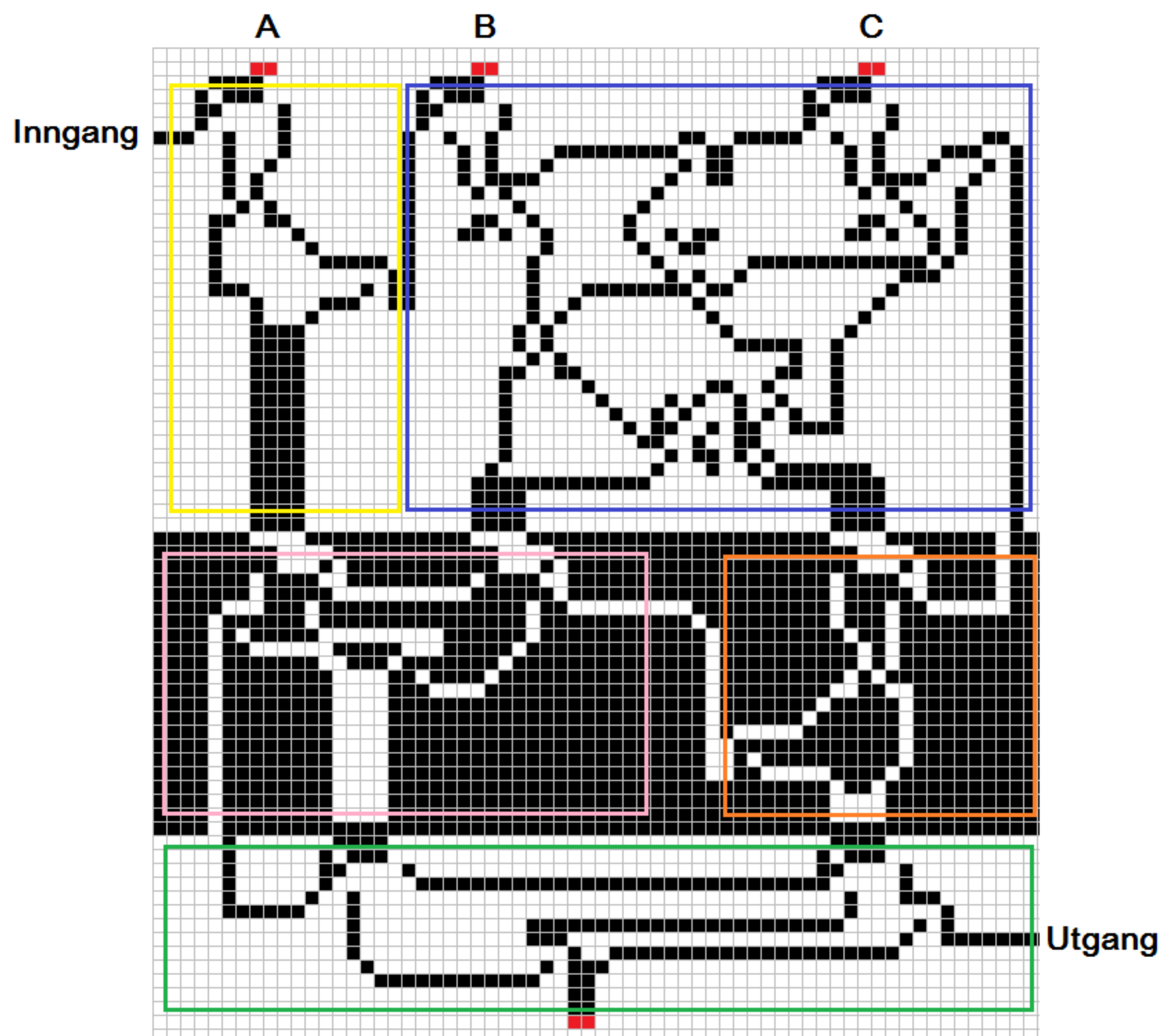
Figur 5.32: Demonstrasjon av dupliserings-port.



Figur 5.33: Demonstrasjon av dupliserings-port.



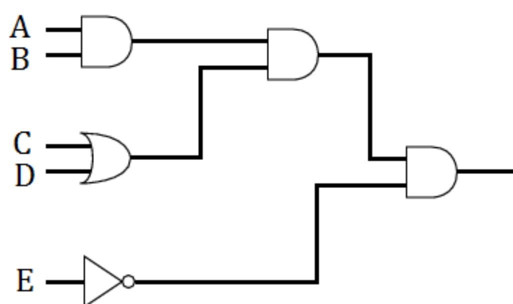
Figur 5.34: Demonstrasjon av kryss-port.



Figur 5.35: Demonstrasjon av kryss-port.

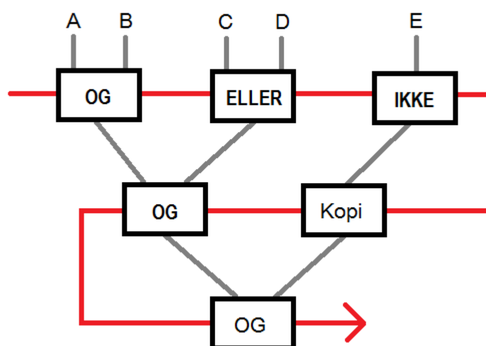
5.7.4 Enkel krets

Nå som alt er på plass, kan vi starte å konstruere kretser med Langtons maur. Vi skal konstruere kretsen i figur 5.36.

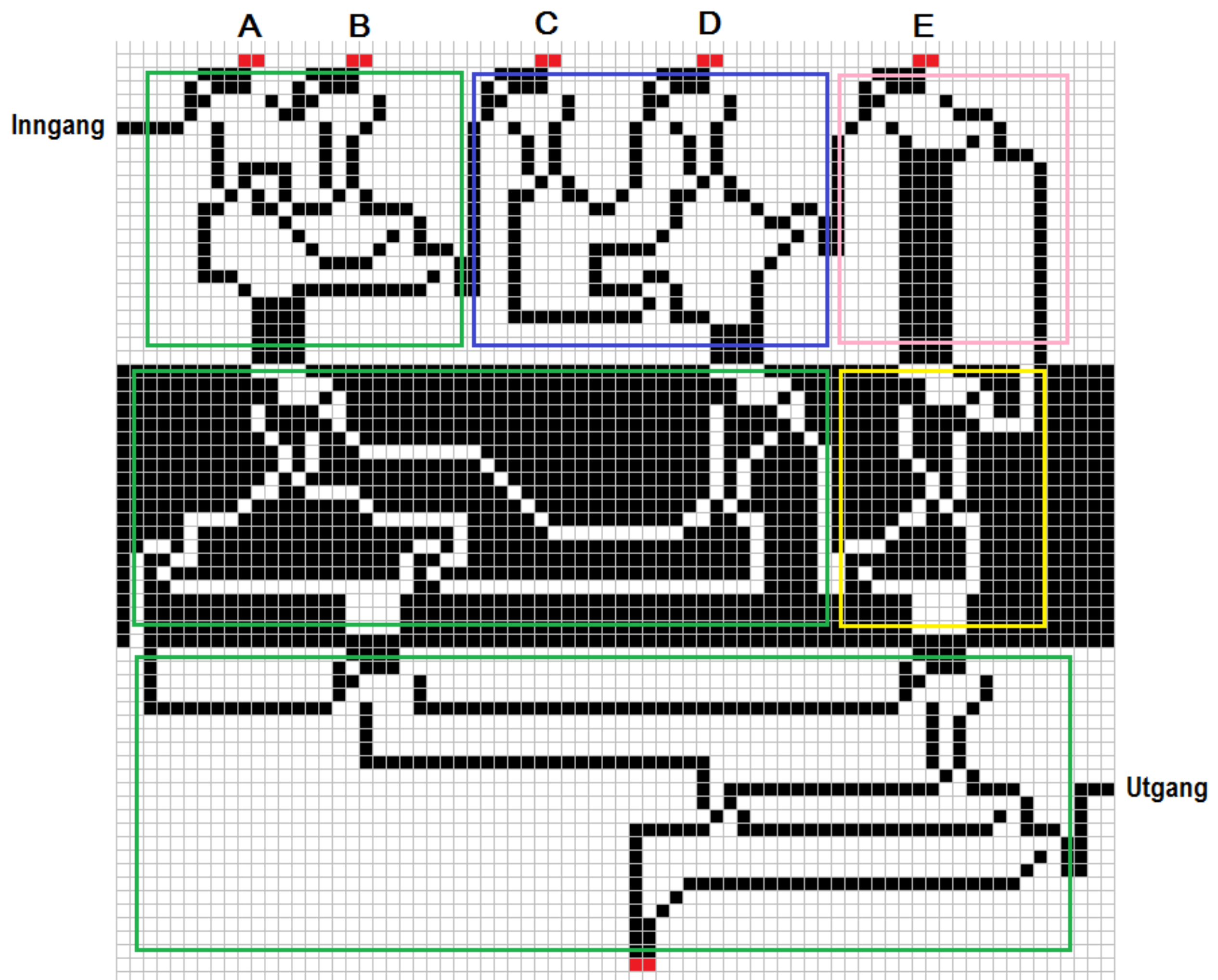


Figur 5.36: Eksempel på en enkel krets.

For å gjøre det enkelt, starter vi med å designe hvordan portene skal kobles sammen. I figur 5.37 har vi lagd en liten modell på hvordan kretsen skal se ut når vi bygger den. Implementasjonen av kretsen er i figur 5.38 hvor de grønne firkantene er OG-porter, blå firkant er en ELLER-port, gul firkant er en kopieringsport og rosa er en IKKE-port.



Figur 5.37: En design av hvordan figur 5.36 kommer til å se ut med Langtons maur porter.



Figur 5.38: Krets med Langtons maur porter av kretsen 5.36.

5.8 Sammendrag

I dette kapittelet har vi sett på hvordan vi kan bruke Langtons maur til å utføre logiske operasjoner. Måten vi har gjort det på er ved å bruke en maur til å gå gjennom alle portene. Dette er ikke så trivielt som å legge portene etter hverandre og under hverandre eller bare å speile om en port og sette porten. I dette kapittelet har vi skissert tre metoder for å løse problemet med porter med inngang på høyre side og utgang venstre side hvor metode 3 var den metoden som vi foretrakk. Fordelen ved å bruke metode 3 er at vi kan speile om en port og deretter bytte farge på stien og bakgrunnen med hverandre. Ulempen er at det kan være forvirrende å finne ut hvilken port som må speiles om og bytte farge på stien. Mens på metode 1 og 2 er fordelen at vi beholder den svarte stien og alle porter fra høyre mot venstre blir unike slik at vi kan lett skille mellom portene. Ulempen er at vi må konstruere portene. Vi støtte også på problemet med å konstruere XOR-porten. Vi klarte ikke å konstruere direkte en XOR-porten med Gajardo sine krysninger, fordi mauren ikke kan huske på inputverdier mauren har lest. Det mauren gjør er bare å følge en sti. Leser mauren SANN på input følger den stien til høyre; leser den USANN følger mauren stien til venstre. Dette medfører at vi må bruke flere porter og iterasjoner for å representere en XOR-port. Ved å introdusere krysning X klarte vi å konstruere XOR-porten. Idéen med krysning X er å bruke det som “hukommelse” for mauren, mens hvis vi skal representere XOR-porten med flere porter, bruker portene outputen som “hukommelse” til å mellom lagre verdier. Fordelen med XOR-porten er at vi slipper å bruke flere porter til å representere XOR og man vil spare flere linjer med iterasjoner. Ulempen er at XOR-porter tar mer plass enn de andre portene slik at høyden på hele iterasjonen der det brukes XOR-port blir høyere. Det betyr at vi må strekke de andre portene mer enn vanlig i samme iterasjon som XOR-porten. I figur 5.38 kan vi se at alle de tre iterasjonene blir ca. like høye, mens i figur 5.35 ser vi at det er mer ujevne høyder på iterasjonene.

Kapittel 6

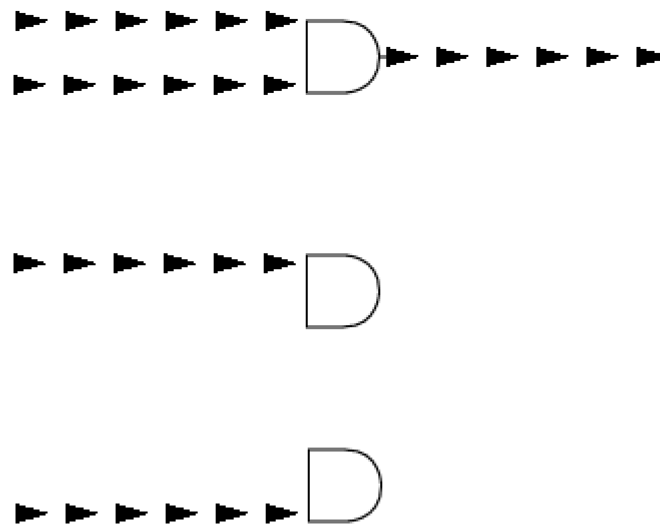
Diskusjon

I forrige kapittel så vi hvordan vi kunne konstruere nye porter ut fra krysningene til Gajardo og bruke portene til å lage kretser. Portene vi konstruerte er for den *vanlige Langtons maur*. Vi introduserte også krysning X som vi benyttet til å lage en XOR-port. Vi støtte også på problemet med at mauren ikke kan huske på inputverdier. Hovedfokuset i dette kapittelet er å finne ut om den generelle Langtons maur kan beregne kretser ved å bruke logiske porter på samme måte som den vanlig Langtons maur gjorde i de forrige kapitlene. For å finne ut om den generelle Langtons maur kan utføre logiske operasjoner skal vi se nærmere på hvordan den vanlige Langtons maur krysser Gajardos krysninger. Når vi sier den generelle Langtons maur i dette kapittelet, mener vi cellulære automater med vilkårlige regelstrenger, men unntatt regelstrenger som består av kun L eller R. Den generelle Langtons maur som tar inn regelstrengen som består av kun L eller R som input er ikke så interessant, fordi mauren går bare rundt i ring enten med klokka eller mot klokka.

6.1 Universelle beregninger

Det finnes mange uløste problemer innenfor cellulære automater. Ett av dem som Wolfram skrev om var "hvor vanlig er universelle beregninger og uavgjørbarhet i cellulære automater" [17]. Tidligere nevnte vi at den vanlige Langtons maur og Game of Life kan utføre universelle beregninger. Metoden som ble brukt til å vise universelle beregninger for Game of Life er å vise at strukturen og mønstrene i Game of Life kan brukes som komponenter til en datamaskin. Paul Rendell som konstruerte en turingmaskin med Game of Life, som vi nevnte tidligere i

kapittel 2, baserte konstruksjonene sine på beviset til John Conway [3]. I 1982 beviste Conway at Game of Life kan utføre universelle beregninger [3]. Idéen som ble brukt for å vise at Game of Life er universell er ganske enkel, men å komme fram til konstruksjonen er en annen sak. Idéen er å konstruere logiske porter til konstruksjonen av digitale systemer. En såkalt *glider* blir brukt til å sende signaler gjennom portene. En glider er en mønster som blir repetert mot en retning i rutenettet. Kort sagt så finnes det et mønster en såkalt "glidergun" som produserer disse gliderene. Dette brukes til å lage nødvendige konstruksjoner for å lage en datamaskin med endelig minne [3].



Figur 6.1: Glider, eksempel av OG-port.

I figur 6.1 er et lite eksempel på en modell for en OG-port på Game of Life hvor de små trekantene mønstrene representerer en glidere. Det som er felles for å vise at Game of Life og Langtons maur er universelle, er at det blir konstruert logiske porter. I beviset for Langtons maur, blir de logiske portene brukt til å simulere endimensjonale cellulære automater. Vi kan for eksempel bruke Langtons maur til å simulere en endimensjonal cellulær automat med input regelstreng 110. Det har blitt bevist at regelstrengen 110 kan utføre universelle beregninger [18]. Hvis et system klarer å simulere et annet system som er universelt, er det selv universelt. Problemet som Wolfram diskuterte er veldig bredt. Vi skal i

dette kapitlet begrense oss til universelle beregninger med den generelle Langtons maur.

6.1.1 Strategi

Vi skal gå ut ifra samme strategi som Gajardo introduserte i [4] ved å lage en konstruksjon slik at den generelle Langtons maur får samme funksjon som den vanlige Langtons maur i artikkel [4]. Det vi må konstruere er:

- Stien for mauren.
- Hvordan mauren skal lese inn en input og hvordan en maur skal skrive ut en output.
- De fire krysningene A, B, C og J.

Hvis vi har de tre punktene for alle regelstrenger for den generelle Langtons maur, kan vi lage de logiske portene og hjelpeportene som trengs for å vise at mauren kan utføre de samme funksjonene som er beskrevet i artikkelen [4]. Fremgangsmåten vi skal bruke, er å gå ut i fra å observere hvordan atferden til den vanlige Langtons maur er når den går gjennom Gajardos sine krysninger. Vi må observere en og en celle i hver krysning for å se hvordan de endrer tilstandene sin.

6.2 Regelstreng LRLRLRLRLR...

Finnes det andre regelstrenger slik at hver celle oppfører seg på samme måte som den vanlige Langtons maur? I denne seksjonen skal vi se på regelstrenger som er på denne formelen $(LR)^+$ og $(RL)^+$. Her betyr + at det som står inne i parentesene kan gjentas en eller flere ganger. Vi vil vise at for alle regelstrenger $(LR)^+$ og $(RL)^+$ kan vi bruke Gajardo sine krysninger til å konstruere de logiske portene som vi konstruerte i kapittel 4. For å gjøre ting mer oversiktlig, skal vi dele tilstandene i en celle i L-type tilstander og R-type tilstander. L-type tilstander er tilstander i en celle som gjør at mauren svinger 90 grader til venstre og deretter går mauren rett frem. R-type tilstand er tilstander i en celle som gjør at mauren svinger 90 grader til høyre og deretter går mauren rett frem.

Eksempel 6.2.1. Anta vi har regelstrengen LLLRRLR da vil tilstandene 0, 1, 2 og 5 være L-type tilstander i en celle og 3, 4 og 6 være R-type tilstander i en celle.

Argumentet er ganske enkel: Vi vet at med disse regelstrengene, kommer alle cellene til å bytte tilstand fra en L-type til en R-type alternerende. Vi vet også at alle cellene til den vanlige Langtons maur bytter tilstand på L-type og R-type tilstand alternerende. Dette fører til at for regelstrengene $(LR)^+$ og $(RL)^+$, kan vi bruke Gajardos sine krysninger. Dette kan gjøres ved å bytte ut tilstander til cellene med svarte tilstander med en R-type tilstand i cellen og alle hvite celler med en L-type tilstand i cellen på alle figurer i kapittel 4 og 5.

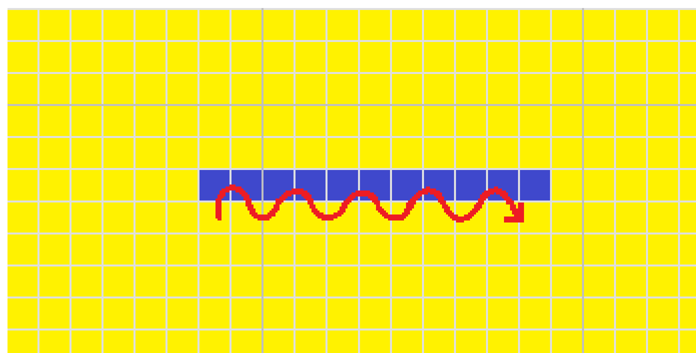
Hva med resten av de mulige regelstrengene? Kan vi konstruere logiske porter med de også? I de neste seksjonene skal vi beskrive hvordan vi konstruerer Gajardos sine krysninger for alle mulige regelstrenger.

6.3 Sti

For den generelle Langtons maur vet vi fra kapittel 2 at mauren tar inn en regelstreng som består av en kombinasjon av en sekvens med L og R. Vi skal vise at vi kan lage en sti til mauren hvis det finnes minst én L og én R i regelstrengen. Vi vet at hvis regelstrengen består av kun L, vil mauren bare gå rundt i ring mot venstre. Det tilsvarende vil skje hvis regelstrengen består av kun R, bare at mauren går i ring mot høyre. Hvis regelstrengen inneholder minst én L og én R, vet vi at cellene på rutenettet kan bytte fra en L-type tilstand til en R-type tilstand og vi vet at alle cellene kan bytte fra en R-type tilstand til en L-type tilstand. Dette kan vi bruke til å lage en sti som er tilsvarende til den vanlige Langtons maur sin sti. Vi vet at cellene på rutenettet til den generelle mauren bytter tilstandene syklisk i samme rekkefølge. Vi lar en L-type tilstand i regelstrengen ha tilstand i og en R-type tilstand i regelstrengen ha tilstand j hvor overgangen fra tilstand i til den neste tilstanden er tilstand j. På bildet under kan vi se hvordan vi kan konstruere stien. For å gjøre det lettere lar vi gul tilstand være i og blå tilstand være j.

Eksempel 6.3.1. Anta vi har regelstrengen RRRRRL. Her ser vi at fra tilstand 5 til tilstand 0 er det overgang fra tilstand en L-type tilstand til en R-type tilstand. Det er kanskje lettere å se det hvis man legger regelstrenger flere ganger etter hverandre som slik: RRRRRLRRRRRLRRRRRLRRR...

Tidligere i denne oppgaven er vi vant til at tilstand 0 har hvit farge og tilstand 1 har svart farge. Fra nå av og utover resten oppgaven lar vi den



Figur 6.2: Sti for den generelle Langtons maur.

hvite tilstanden være tilstand i og den svarte tilstanden være tilstand j hvor i er en L-type tilstand som har overgang til en R-type tilstand.

6.4 Input og output

Fra den vanlige Langtons maur husker vi at USANN er definert som to hvite celler ved siden av hverandre og SANN definert som to svarte celler ved siden av hverandre for input og output. For den generelle Langtons maur bestemte vi oss for å la alle celler i rutenettet med L-type tilstand være USANN og alle celler i rutenettet med R-type tilstander i regelstrengen være SANN slik at vi får to disjunkte mengder hvor den ene mengden er mengden med alle L-type tilstander og den andre mengden med R-type tilstander.

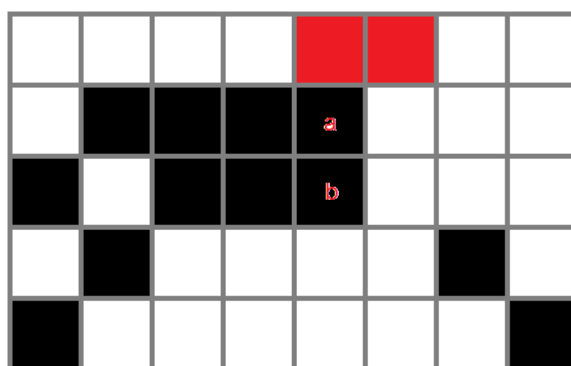
Eksempel 6.4.1. Anta regelstrengen LLLRLRL. Da vil tilstandene 0, 1, 2, 4 og 6 i en celle være de tilstandene som kan representere de USANNE inputverdiene. Og tilstandene 3 og 5 i en celle vil være de tilstandene som kan representere SANNE inputverdier og outputverdier.

Grunnen til at vi valgte å definere inputen og outputen slik er fordi det kan bli problematisk hvis vi for eksempel lar begge tilstandene i og j i en celle være L-type tilstander og i skal representere USANN mens j representere SANN vil mauren gå til venstre etter å ha hentet inputen i eller j. Mauren vil da følge den samme stien for begge inputer i og j som blir problematiske. En ting som man må legge merke til er når den vanlige Langtons maur leser inn en input er det noen celler som blir endret mer enn en gang. Vi må lage en generell konstruksjon slik at den generelle Langtons maur kan hente en input. La oss se på hvordan

den vanlige Langtons maur endrer tilstandene på cellene når den henter input:

Celle	Endring av tilstand når input er SANN
a	R -> L
b	R -> L

Tabell 6.1: Endringene i cellene for å lese en SANN input.

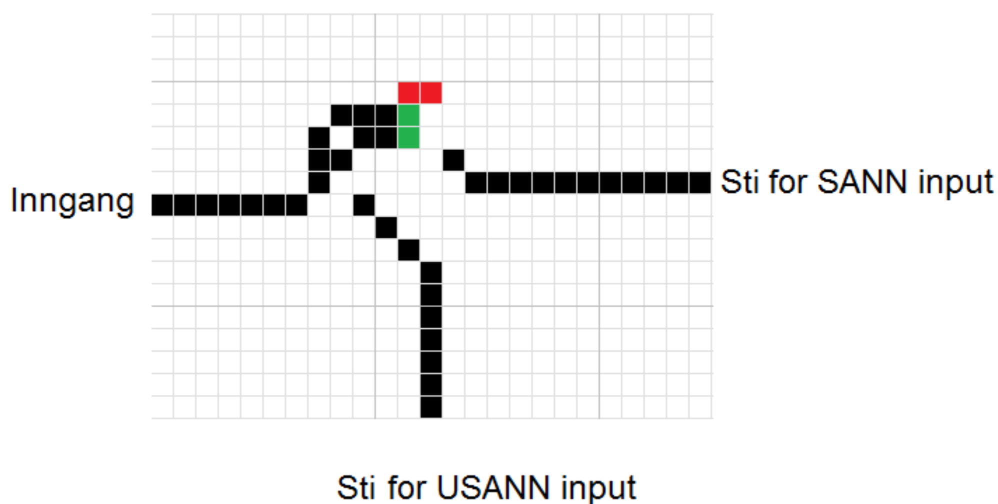


Figur 6.3: Lese input for den generelle Langtons maur.

Eksempel 6.4.2. Når det står L-> R-> L i tabellen for en celle *i* betyr det at mauren har gått på cellen *i* tre ganger. Første gang når mauren går på celle *i* har den en L-type tilstand, andre gang en R-type tilstand og tredje gang en L-type tilstand. Når det står - så betyr det at cellen er urørt når mauren bruker krysningen.

For input USANN endrer hver tilstand seg maks en gang og vi kan bruke den vanlige stien for å lede mauren gjennom, men for input SANN, ser vi at cellene a og b endrer seg to ganger og ender opp med en R-type tilstand. Dette kan vi lett fikse ved å la celle a og b ha starts tilstander som er en R-type tilstand som endrer seg til en L-type tilstand når den økes. Vi vet at det finnes en tilstand som har overgang fra en R-type tilstand til en L-type tilstand hvis det det finnes minst én L og én R i regelstrengen. I figuren 6.4 kan vi se hvordan det kan gjøres ved å la de grønne cellene være de cellene som har en tilstand slik at hvis mauren går på cellen blir den nye tilstanden til cellen en L-type tilstand.

Eksempel 6.4.3. Anta vi har regelstrengene LLLLRLL og RLLL. Da vil den grønne cellen ha tilstand 4 (LLLLRLL) og 0 (RLLL) fordi de er de eneste R-type tilstand som blir til en L-type tilstand når tilstanden øker med en. Anta regelstrengen RLLRLRL da kan vi ha flere muligheter. De grønne cellene kan ha tilstandene 0, 3 og 5.



Figur 6.4: Lese input for den generelle Langtons maur.

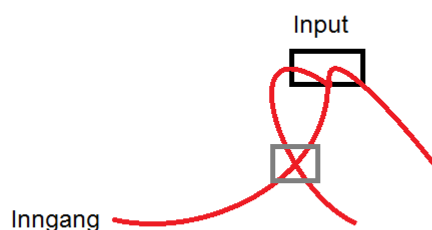
For den generelle Langtons maur må vi også definere outputen på en annen måte. Fra den vanlige Langtons maur husker vi at vi lot output cellene ha standard tilstand som USANN. For den generelle Langtons maur kan vi ikke bare velge en hvilket som helst tilstand i mengden av L-type tilstander. Vi må la output cellene ha en L-type tilstand *i* slik at tilstand *i+1* er en tilstand i mengden av R-type tilstander. Grunnen er fordi når mauren skal ned og skrive ut SANN i output cellene så går mauren ned og endrer de to output cellene kun en gang. Derfor må vi la output cellene være L-type tilstander som blir til R-type tilstander når de endres.

Eksempel 6.4.4. Anta vi har regelstrengen LLLRRRLR. De tilstandene som vi kan bruke som standard output celler er 2 og 6. Fordi når mauren går ned og endrer tilstandene til output cellene, får cellene de nye tilstandene 3 og 7 som er SANN. Dette gjelder for kun porter som har inngang på venstreside og utgang på høyre side. For porter med inngang på høyreside og utgang venstreside må vi velge tilstand 5 og 7. Dette er fordi portene med inngang på høyreside og utgang har standard

output SANN og skal byttes til USANN hvis mauren kommer og endrer outputen.

6.5 Krysning A

Som vi husker fra det forrige kapitlet, brukes krysning A til å krysse en sti som mauren har allerede gått på. Dette tilfelle skjer når inputen er USANN (hvit) og mauren må krysse stien som mauren brukte til å hente inputen.



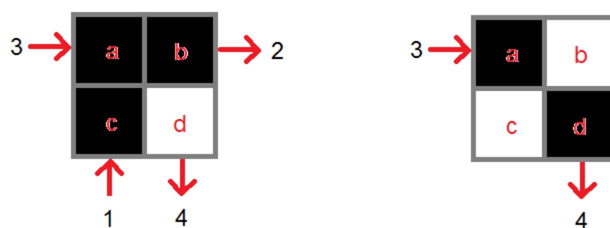
Figur 6.5: Illustrasjon av hvordan den vanlige Langtons maur utfører krysning A.

Hvis vi ser på hvordan den vanlige Langtons maur krysser krysning A i figur 6.6 er det ikke alltid like lett å se at vi fra 1 til 2 og fra 3 til 4 i Gajardos krysning A blir cellene besøkt maks en gang hver vei. Når mauren går fra 1 til 2 endrer tilstandene til cellene b, c og d en gang og cellene kommer til å se ut som det andre bildet i figur 6.6. På vei tilbake når mauren går fra 3 til 4 endrer også cellene maks en gang. Totalt er det to celler som endrer tilstand to ganger. Tabell 6.2 viser hvordan cellene i krysning A endrer seg hvis mauren skal inn inngang 1 og deretter inn inngang 3.

Celle	Endring av tilstand
a	R
b	R
c	R -> L
d	L -> R

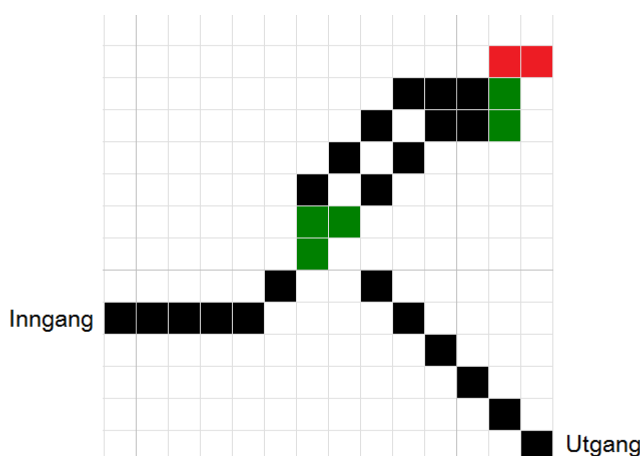
Tabell 6.2: Endringene i cellene for krysning A.

Å konstruere denne krysningen kan være forskjellig fra regelstreng til regelstreng. Siden det er minst én L og én R i regelstrengen vet vi at det



Figur 6.6: Illustrasjon av hvordan den vanlige Langtons maur utfører krysning A.

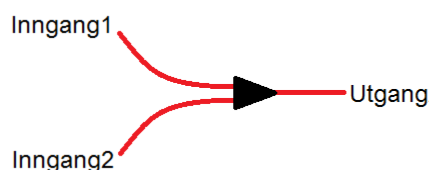
finnes en tilstand *i* (som er en L-type tilstand) slik at tilstand *i+1* er en R-type tilstand og motsatt. Dette kan vi utnytte til å konstruere krysning A for alle regelstrenger der det er minst én L og én R. På denne måten kan vi lage krysning A for den generelle Langtons maur. I figur 6.7 ser vi hvordan krysning A blir for den generelle Langtons maur hvor den grønne cellen er en R-type tilstand som endrer sin tilstand til en L-type tilstand når mauren går på den.



Figur 6.7: Krysning A for den generelle Langtons maur.

6.6 Krysning J

Krysning J brukes til å slå sammen to stier til en sti. Det vil si at mauren kommer inn enten fra inngang 1 eller 2 og kommer ut fra samme utgangen.



Figur 6.8: Illustrasjon av hvordan den vanlige Langtons maur utfører krysning J.

Når den vanlige Langtons maur går inn i inngang 1 og ut utgangen, beveger mauren slik at hver celle endrer sin tilstand maks en gang. Hvor celle a, b, f og i fungerer som en sti som går fra 1 til 3. Den delen er ganske greit å konstruere. Problemet er når mauren kommer fra inngang 2 da begynner mauren å gå på to celle fler enn en gang før den kommer ut utgangen. Når mauren kommer inn i inngang 2, går mauren i en ring rundt cellene g , h, e, d, g og h som vi kan se i figur 6.9, før den går videre. Vi kan se på tabellen 6.3 hvordan cellene endrer seg.

Celle	Endring av tilstand fra inngang 1	Endring av tilstand fra inngang 2
a	R	-
b	R	-
c	-	-
d	L	L
e	L	L
f	R	-
g	-	R -> L
h	L	L -> R
i	R	-

Tabell 6.3: Endringene i cellene for krysning J.

Vi konstruerte krysning J for den generelle Langtons maur slik som i figur 6.10. Vi måtte utvide plassen slik at hvis det er en regelstreng med mange R, kommer mauren til å gå rundt i ring før mauren går ut utgangen. I figuren har vi markert med en grå boks hvor mauren går i ring før den går videre.

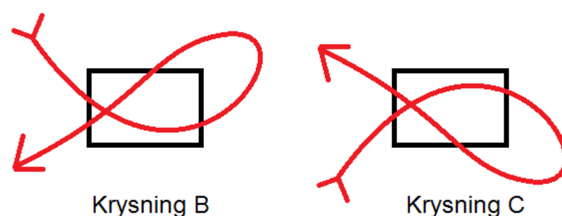
Figur 6.9: Illustrasjon av hvordan den vanlige Langtons maur utfører krysning J.

Figur 6.10: Krysning J for den generelle Langtons maur.

6.7 Krysning B og C

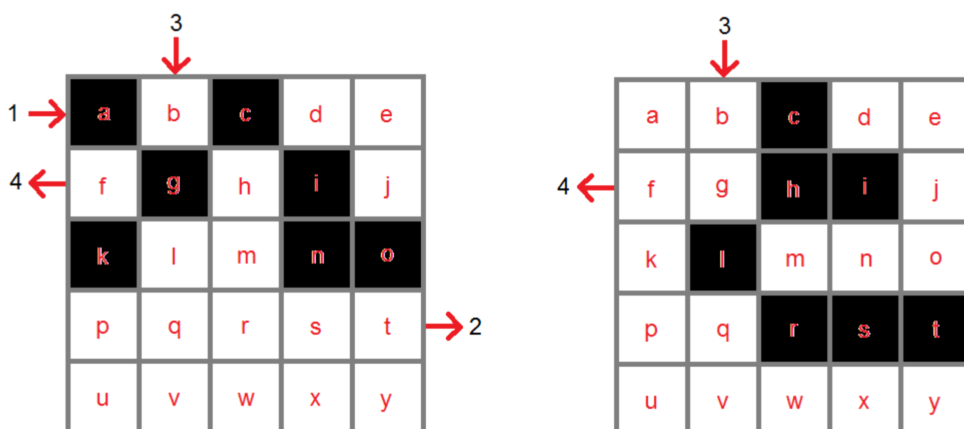
Nå som vi har klart å lage krysning A og J for alle regelstrenger som inneholder minst én L og én R, gjenstår det å lage både krysning B og C som er den vanskeligste utfordringen av alle krysningene. I krysning B og C er det slik at mauren ikke endrer på sporet sitt innenfor krysningen bare en gang, men kan være opp mot fire ganger. Krysning B og C er viktig for å få til kryss-porten. I kapittel 4 konstruerte vi de andre portene ved å kun bruke krysning B fordi vi brukte krysning B for å enten gå inn 1 og ut 2 eller gå inn 3 og ut 4. Vi brukte aldri det tilfellet at mauren senere skal gjenbruke krysningen. Dette problemet oppstår når vi skal lage kryss-porten, da må vi ta hensyn til at krysningen skal brukes igjen. Forskjellen på krysning B og C er at de er speilvendt av hverandre. Det er ikke lett å se at man trenger både krysning B og C for kryss-porten og man tenker at det ikke har noe å si. Hvis man prøver å konstruere kryss-porten helt fra bunnen, kan man lettere se at det er nødvendig med både krysning B og C. I figur 6.11 illustrerer vi hvordan maurene beveger seg

i de to krysningene. Uansett hvor mye vi roterer krysning B og C, blir de aldri lik hverandre.



Figur 6.11: Forskjellen på krysning B og krysning C.

Fra kapittel 5 vet vi at vi kan rotere en krysning med og mot klokka så mye vi vil, men vi kan ikke speile om en port uten å bytte farge på stien og bakgrunnen. Før vi setter i gang og konstruerer krysning B og C for den generelle Langtons maur, skal vi se på hvordan cellene på krysning B og C endrer seg når mauren går gjennom krysningene. I den første tabellen 6.4 ser vi endringer av celler i krysning B hvor den første kolonnen beskriver endringene når mauren bruker krysningen to ganger. Mens i den andre kolonnen beskriver endringen av cellene for inngang 3 og utgang 4. I tabell 6.5 er det beskrivelsen for krysning C med samme oppsett som tabellen på krysning B.



Figur 6.12: Hvordan den vanlige Langtons maur utfører krysning B. Det andre bildet viser hvordan cellene ser ut etter mauren har gått inn i inngang 1.

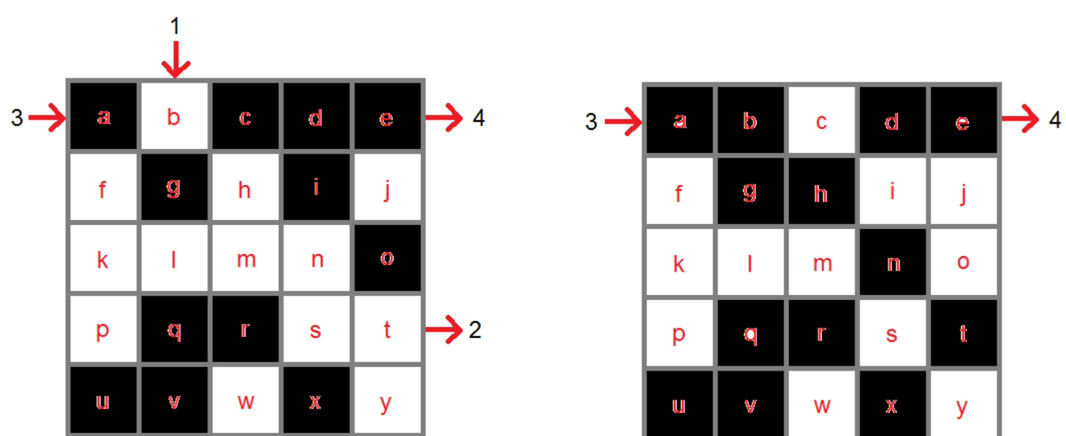
Celle	Inn 1 og ut 2, deretter inn 3 og ut 4	Inn 3 og ut 4
a	R -> L	-
b	L -> R	L
c	R -> L	R -> L
d	-	L
e	-	-
f	L -> R -> L	L
g	R -> L -> R -> L	-
h	L -> R	L -> R -> L
i	-	R -> L -> R
j	-	R
k	R -> L -> L	R
l	L -> R -> L -> R -> L	L
m	L -> R	L -> R -> L
n	R	R -> L -> R
o	R	-
p	L	-
q	L	L -> R
r	L	L -> R -> L
s	L	L
t	L	-
u	-	-
v	-	L
w	-	L
x	-	-
y	-	-

Tabell 6.4: Endringer i cellene for krysning B.

Vi kan allerede se at dette kan bli problematisk for den generelle Langtons maur. De cellene som endrer seg opp til flere ganger er der problemet ligger. Når mauren går på sporet sitt flere ganger blir det vanskelig å finne ut hvordan man kan lage en generell metode for å lage krysningen. Vi kan observere at den lengste sekvensen med endringer for både krysning B og C er LRLRL. Vi ser også at sekvensen til alle de andre endingene er en subsekvens av LRLRL. Det betyr at hvis vi har en regelstreng *s* hvor *s* kan endre tilstandene sine i rekkefølgen L-> R-> L-> R-> L i cellene sine kan den bruke Gajardos sine krysning B og C. Vi referer dette for tabellmetoden.

Celle	Inn 1 og ut 2, deretter inn 3 og ut 4	Inn 3 og ut 4
a	R	R
b	L	-
c	R	-
d	R	R
e	R	R
f	L	L -> R
g	R	R -> L -> R -> L
h	L -> R	L -> R -> L -> R
i	R -> L	R -> L
j	L	L
k	-	L -> R -> L
l	L	L -> R -> L -> R -> L
m	L	L -> R -> L -> R -> L
n	L	L -> R -> L
o	R	-
p	-	L -> R -> L
q	-	R -> L -> R
r	-	R -> L -> R
s	-	L -> R -> L
t	L	-
u	-	R
v	-	R -> L
w	-	L -> R
x	-	R
y	-	-

Tabell 6.5: Endringer i cellene for krysning C.



Figur 6.13: Hvordan den vanlige Langtons maur utfører krysning C. Det andre bildet viser hvordan cellene ser ut etter mauren har gått inn i inngang 1.

Eksempel 6.7.1. Anta at vi har laget en ny krysning Z hvor endringer til cellene er beskrevet som i tabell 6.6, ser vi at sekvens LRLR er den lengste sekvensen, men den LRLR har ikke alle de andre sekvensene som subsekvens. Den har ikke sekvensen LLL som subsekvens. Vi kan konkludere med at hvis en regelstreng kan endre tilstandene i cellen i både rekkefølgen LRLR og LLL, kan mauren med regelstrengen bruke utføre krysning Z.

Celler	Krysning Z
a	L -> R -> L -> R
b	L -> R
c	R -> L
d	L -> L -> L
e	L -> R
f	R -> L

Tabell 6.6: Eksempel 6.7.1

6.8 Konstruksjon av nye krysninger

Så langt har vi klart å lage krysning A og J for den generelle Langtons maur mens krysning B og C er fortsatt vanskelig. Selv om vi ikke klarer å vise en generell metode for å lage krysning B og C for den generelle Langtons maur, kan vi eksperimentere oss frem til krysningene for hånd. Vi tar utgangspunkt i Gajardo sine krysninger og simulerer den generelle Langtons mauren med en gitt regelstreng. Deretter endrer vi de cellene som trengs for at krysningen skal få samme funksjon som Gajardo sine krysninger. Noen regelstrenger tok mer tid enn andre spesielt de som har en lang regelstreng.

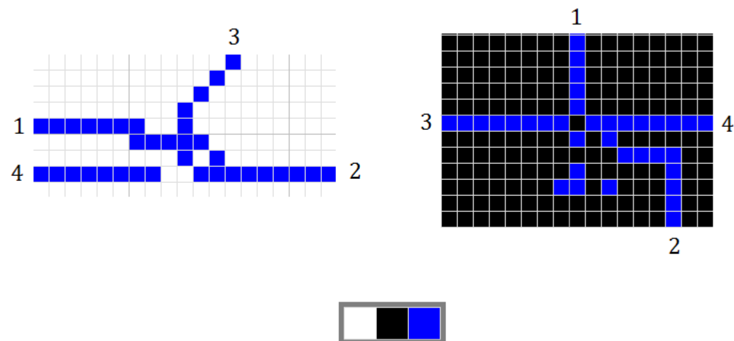
6.9 Resultater av konstruksjoner av krysninger

I denne seksjonen skal vi bruke metoden som er beskrevet ovenfor til å konstruere krysning B og C for alle regelstrenger opp til lengde fem. For hver regelstreng er det $2^r - 2$ kombinasjoner hvor r er lengden av regelstrengen. Fordi det er to mulige tilstander på hver bokstav i regelstrengen og vi må ta vekk de regelstrengene som inneholder kun L eller R som blir å trekke vekk to. Vi må også huske på at hver celle bytter tilstand syklisk som gjør at, hvis vi for eksempel klarer å lage en krysning

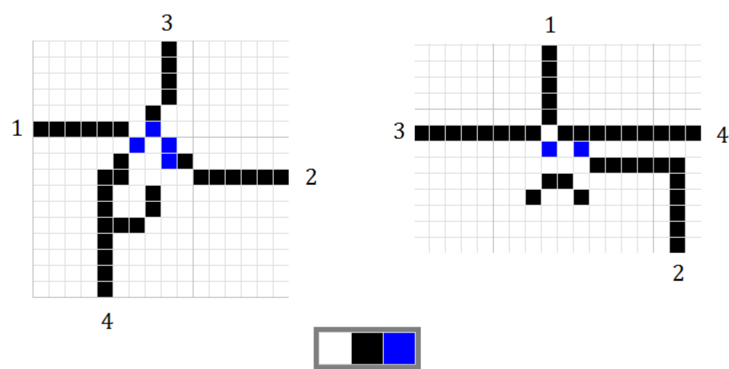
B og C for regelstrengen LLR, kan LRL og RLL også bruke den samme krysningene. Fordi LLR, LRL og RLL bytter tilstand i lik rekkefølge. Det vi trenger å gjøre er å justere alle tilstandene med en konstant slik at de kan utføre det samme.

- For alle regelstrenger med lengde to har vi LR og RL. For RL kan vi bare bytte om på tilstandene på krysning B og C. Svart til hvit og motsatt. I tillegg må vi initialisere hele rutenettet med svart tilstand og bruke hvit tilstand som en sti.
- For alle regelstrenger med lengde tre har vi LLR og LRR som dekker alle de andre regelstrengene på lengde tre. På samme måte som LLR dekker LRL og RLL, dekker LRR regelstrengen RLR og RRL.
- For alle regelstrenger med lengde fire trenger vi å lage krysning B og C for LLRR eller RRLL, og vi må lage for LLLR og LRRR. Resten av kombinasjonene for regelstrenger med lengde fire dekkes på seksjon 6.2.
- For alle regelstrenger med lengde fem, kan LRLRL bruke samme krysning som den vanlige Langtons maur sine krysninger. Det som gjenstår er å lage for LLRRR, LLLRR, RLRLR, LRRRR og LLLLR.

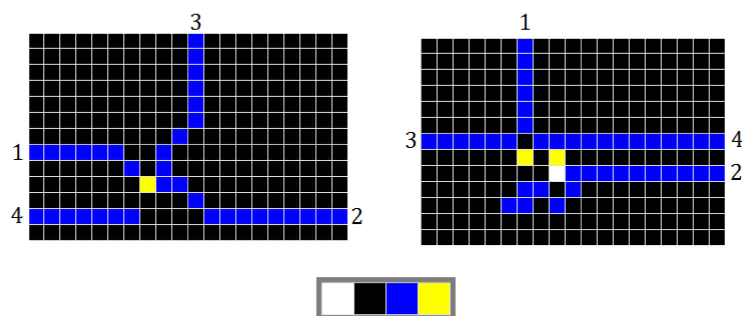
Numrene på utgangen er de samme som er beskrivet i kapittel 4, "Hvis mauren går inn 1, går den ut 2, senere hvis mauren går inn 3, går den ut 4. Eller hvis mauren går inn 3, går den ut 4". Det første bildet på hver figur er krysning B og det andre bildet er krysning C mens baren med farger under begge bildene er fargen på tilstandene. Hvor fargen lengst mot venstre er tilstand 0 og økende med 1 mot høyre. B og C krysningene i figurene har vi konstruert ved bruk av metoden som er beskrevet i 6.8 og det finnes garantert andre måter å lage disse krysningene på.



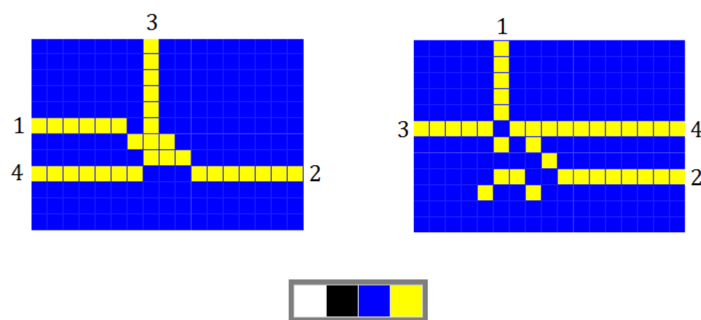
Figur 6.14: Regelstreng LLR



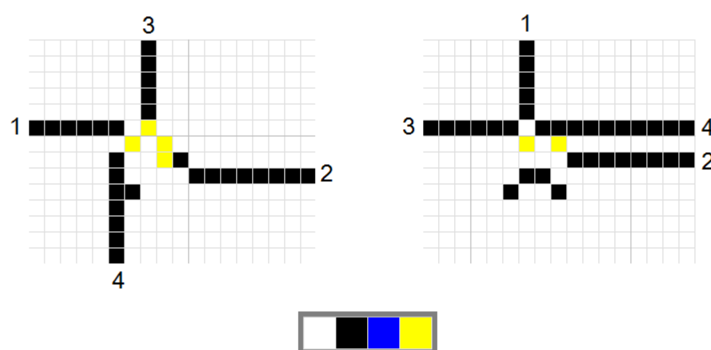
Figur 6.15: Regelstreng LRR



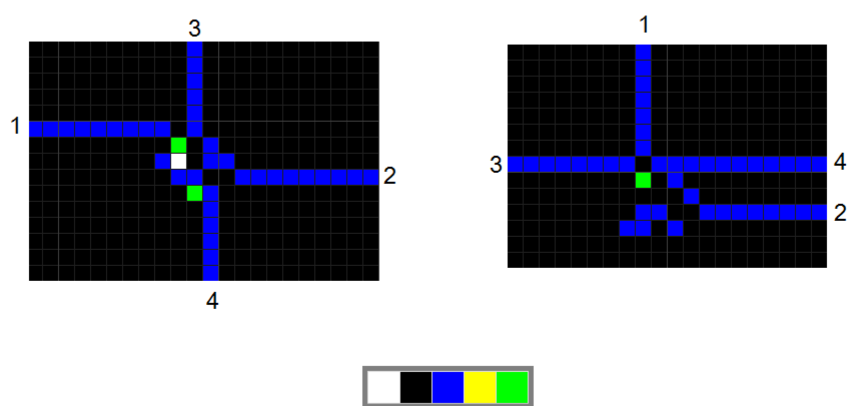
Figur 6.16: Regelstreng LLRR



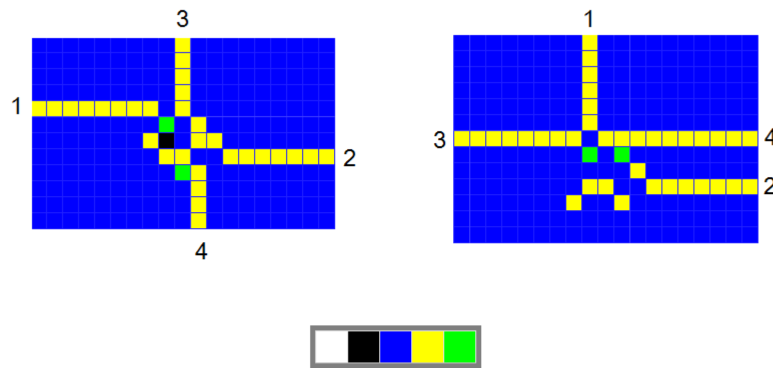
Figur 6.17: Regelstreng LLLR



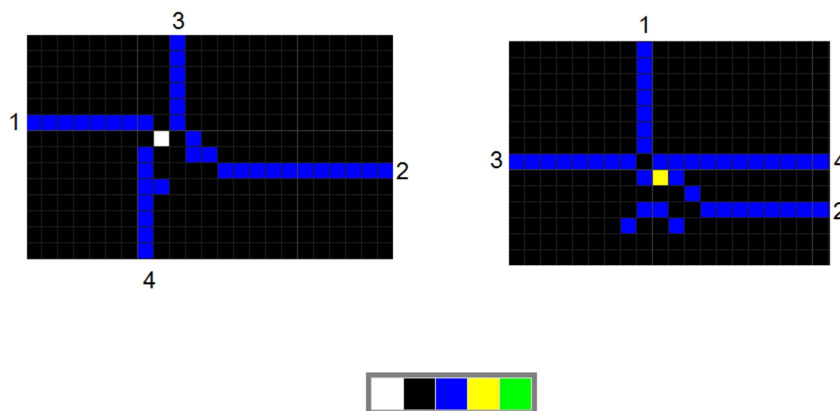
Figur 6.18: Regelstreng LRRR



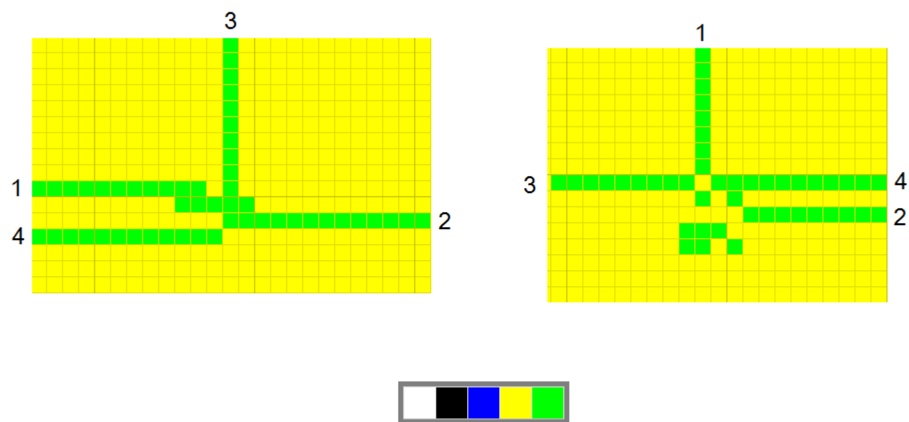
Figur 6.19: Regelstreng LLRRR



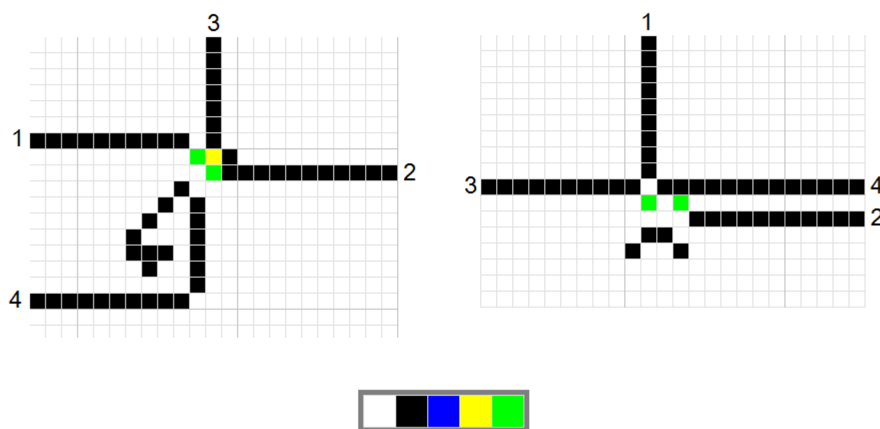
Figur 6.20: Regelstreng LLLRR



Figur 6.21: Regelstreng RLRLR



Figur 6.22: Regelstreng LLLLR



Figur 6.23: Regelstreng LRRRR

6.10 Kardioiden er universell

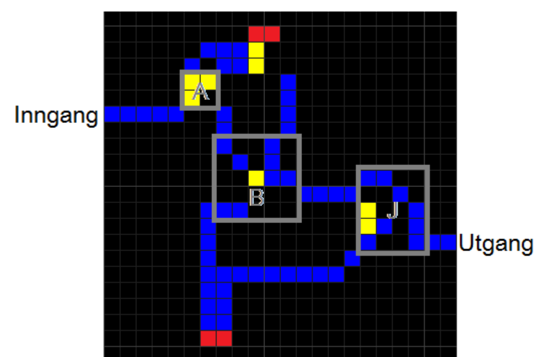
I forrige seksjon har vi sett på et par regelstrenger som er universelle. I denne seksjonen skal vi demonstrere at kardioiden er universell slik som Gajardo i [4] demonstrerte at den vanlige Langtons maur er universell. Fra kapittel 2 husker vi at mønstrene til sporet til den generelle Langtons med regelstreng LLRR som input lager en kardioide. Vi skal nå bruke det vi har beskrevet i dette kapittelet for å lage OG, IKKE og hjelpeportene.



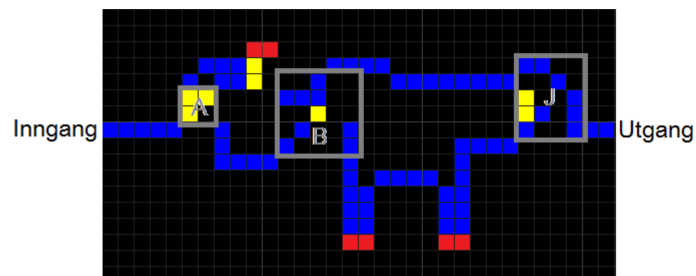
Figur 6.24: LLRR OG-port



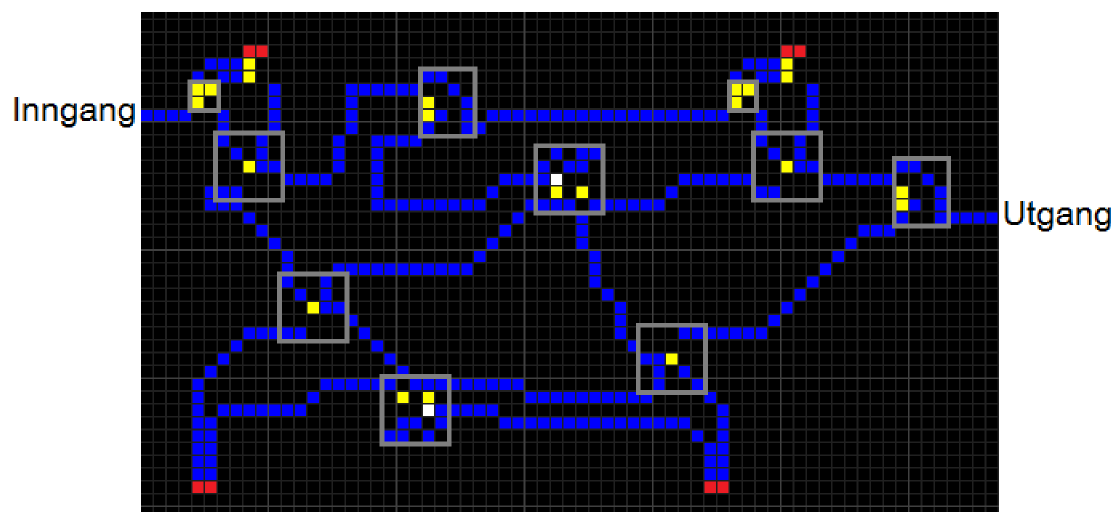
Figur 6.25: LLRR IKKE-port



Figur 6.26: LLRR kopierings-port



Figur 6.27: LLRR dupliserings-port



Figur 6.28: LLRR kryss-port

6.11 Sammendrag

I dette kapitlet er det sentrale spørsmålet, "kan den generelle Langtons maur utføre universelle beregninger?". Det finnes flere metoder for å finne ut om en cellulær automat kan utføre universelle beregninger. En metoden er å identifisere mønstrene til cellulære automater og sjekke om de kan fungere som komponenter til en datamaskin. En annen metode er å vise at cellulære automaten kan simulere en hvilken som helst cellulær automat [17]. Den metoden som er beskrevet i [4] innebærer å bruke den vanlige Langtons maur til å simulere en vilkårlig endimensjonal cellulære automater. Vår metode for å vise at den generelle Langtons maur kan utføre universelle beregninger er å konstruere det samme som Gajardo konstruerte i [4] for den generelle Langtons maur. For at den generelle Langtons maur skal oppnå dette, må vi kunne lage de fire krysningene. Ved å konstruere de fire krysningene medfører at den generelle Langtons maur kan gjøre de samme funksjonene som den vanlige Langtons maur gjorde i kapittel 4 og 5. Vi definerte alle L-type tilstander for USANN og alle R-type tilstander for SANN. Vi fant ingen andre måter å representere SANN og USANN på, for den generelle Langtons maur. For å kunne lage porter med den generelle Langtons maur måtte vi også hente inputen og skrive outputen på en annen måte som vi beskrev i seksjon 6.4. Vi klarte ikke å finne et generelt mønster for å lage krysningen B og C, på grunn av at mauren besøker noen celler flere ganger før den har krysset krysningen. I tillegg beveger hver av de generelle Langtons maurene forskjellig for hver regelstreng som input. Vi prøvde oss fram ved å eksperimentere frem til de krysningene for noen av regelstrengene. Fra å eksperimentere observerte vi at krysning B ble mer komplisert for lengre regelstrenger og at noen celler i krysningen blir besøkt opp til flere ganger. Mens krysning C ser omtrent lik ut som Gajardo sin krysning C. Under konstruksjonen var de regelstrengene som inneholdt mange L og få av R, og de som inneholdt få L og mange R en utfordring å konstruere, som regelstrengene LLLLR og LRRRR. Vi har ikke klart å finne en god måte å vise at alle regelstrenger kan utføre universelle beregninger og vi har heller ikke klart å finne en regelstreng som ikke kan utføre universell beregninger (bortsett fra regelstrenger med kun L eller R). Dette styrker hypotesen om at alle regelstrenger er universelle. Ved å bruke tabellmetoden på hver av de nye krysningene B og C vi konstruerer finner vi også ut hvilken regelstrenger som kan bruke de. Dette gjøres ved å se på sekvensen med endringer til den cellen som endrer tilstand flest ganger i krysningen, deretter sjekker vi om denne sekvensen har alle de andre cellene (i samme krysningen) sine endringer

som subsekvenser slik som vi gjorde på i slutten av seksjon 6.7. På denne måten får vi flere og flere regelstrenger som er universelle og vi kommer nærmere og nærmere alle mulige regelstrenger som inneholder minst én L og én R. Selv om vi støtte på vanskeligheter med å vise at den generelle Langtons maur med alle regelstrenger med minst én L og én R kan utføre universelle beregninger. Vi har en mistanke om at alle regelstrengene kan utføre universelle beregninger. For alle regelstrenger hittil har vi klart å lage krysning B og C for.

Kapittel 7

Konklusjon

I denne oppgaven har vi sett hvordan vi konstruerer logiske porter med Langtons maur og den generelle Langtons maur. Vi så på hvilken porter som var mulig å lage dirkete og hvilken porter vi måtte bruke flere iterasjoner for å representere. Da støtte vi på problemer med den logiske porten XOR som vi ikke klarte å konstruere direkte, men måtte bruke de andre logiske portene til å lage den. Senere introduserte vi en ny krysning, krysning X, som gjorde det mulig å konstruere XOR-porten direkte. Videre brukte vi de logiske portene til å lage en enkel konstruksjon av kretser som ikke er så triviell som man tror det er. Å finne en felles krysning for alle regelstrenger som input for den generelle Langtons maur var mer komplisert enn å konstruere enn å finne krysninger for hver enkelt av regelstrengene. Det var denne måten vi prøvde å bevise at for alle regelstrenger som input så kan mauren utføre universelle beregninger. Selv om vi ikke lykkes med å vise at alle regelstrenger er universelle, klarte vi å vise for alle regelstrenger opp til strenglengde fem og vi har en mistanke om at alle regelstrengen som input kan utføre universelle beregninger.

7.1 Fremtidig arbeid

Fra denne oppgaven gjenstår det fortsatt å bevise at den generelle Langtons maur kan utføre universelle beregninger. Ved å vise at vi kan lage krysning B og C for den generelle Langtons maur. Eventuelt motbevise ved å finne en regelstreng som vi ikke kan lage enten krysning B eller C. Vi har tidligere nevnt at det er lite som har blitt bevist om Langtons maur sitt atferd. Andre ting vi ikke har sett veldig mye av i denne oppgaven var krysning X. Vi fikk ikke sett på begrensninger og

nye muligheter vi kan konstruere med krysning X.

Bibliografi

[1] Basic of automata theory — Stanford. http://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html#fsmturing. [Online; tilgang 4.februar 2016].

[2] Cellular automata — Stanford. http://plato.stanford.edu/entries/cellular-automata/. [Online; tilgang 15.februar 2016].

[3] Berlekamp, E. R.; Conway, J. H. *What Is Life? Ch. 25 in Winning Ways for Your Mathematical Plays, Vol. 2: Games in Particular*. London: Academic Press, 1982.

[4] A. Gajardo, A. Moreira, and E. Goles. Complexity of langton's ant. Final version in Discrete Applied Mathematics, Vol. 117, Issue 1-3, pp. 41-50 (2002), 2003.

[5] D. Gale. *The Industrious Ant*. The Mathematical Intelligencer, 1991.

[6] P. Rendell. *A Universal Turing Machine in Conway's Game of Life*. Department of Computer Science University of the West of England.

[7] M. Sipser. *Introduction to the Theory of Computation*. Thomson Course Technology, 2006.

[8] Tim Taylor. Weak artificial life versus strong artificial life. http://www.tim-taylor.com/papers/thesis/html/node34.html. [Online; tilgang 25.februar 2016].

[9] Wikipedia. Artificial life — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Artificial_life, 2016. [Online; tilgang 25.februar 2016].

[10] Wikipedia. Artificial neural network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Artificial_neural_network, 2016. [Online; tilgang 25.februar 2016].

[11] Wikipedia. Boolean algebra — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Boolean_algebra_(structure), 2016. [Online; tilgang 24.april 2016].

[12] Wikipedia. Cellular automaton — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Cellular_automaton, 2016. [Online; tilgang 24.mars 2016].

[13] Wikipedia. Game of life — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life, 2016. [Online; tilgang 24.april 2016].

[14] Wikipedia. Game of life — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Complexity_class, 2016. [Online; tilgang 25.april 2016].

[15] Wikipedia. Langton's ant — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Langton%27s_ant, 2016. [Online; tilgang 5.februar 2016].

[16] Wikipedia. Logic gates — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Logic_gate, 2016. [Online; tilgang 14.mars 2016].

[17] S. Wolfram. Twenty problems in the theory of cellular automata. *Physica Scripta*, 1985(T9):170, 1985.

[18] S. Wolfram. *A New Kind of Science*. Wolfram Media Inc., Champaign, Ilinois, US, United States, 2002.